
Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e Informática

Bernardo Francisco Peralta Pires da Silva

UM PROCESSADOR COM ARQUITECTURA MIPS PARA ENSINO

Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e Informática

Bernardo Francisco Peralta Pires da Silva

UM PROCESSADOR COM ARQUITECTURA MIPS PARA ENSINO

Dissertação apresentada à Universidade de Aveiro, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de Prof. Doutora Iouliia Skliarova, professora auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

“Aqueles que passam por nós não vão sós. Não nos deixam sós. Deixam um pouco de si… Levam um pouco de nós…”
Antoine de Sant. Exopery

o júri

presidente

Doutor António Ferreira Pereira de Melo
Professor Catedrático da Universidade de Aveiro

vogais

Doutora Iouliia Skliarova (Orientadora)
Professor Auxiliar da Universidade de Aveiro

Doutor Joaquim José Castro Ferreira
Professor Adjunto do Departamento de Engenharia Informática da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

agradecimentos

Nas entre linhas deste documento escondem-se alegrias, sorrisos, tristezas, desânimos, novas amizades e grandes conquistas. Guardo lembranças e saudades de todos aqueles que estiveram presentes ao longo dos momentos de compõem o meu percurso académico.

Antes de mais, um especial obrigado à minha orientadora, Prof. Iouliia Skliarova, pela proposta da dissertação, prontidão de ajuda, acompanhamento e aconselhamento sobre o trabalho realizado, assim como pela revisão dos textos apresentados neste documento que demonstram os objectivos atingidos sobre sua orientação.

Agradeço ainda ao Prof. Valeri Skliarov, Prof. Arnaldo Oliveira, Mestre Manuel Almeida e Eng. Bruno Pimentel pelo tempo disponibilizado, contribuições e ajudas.

Ao Eng. João Lopes pelas horas dedicadas a ouvir as minhas dúvidas, pelas conversas construtivas e conselhos que me guiaram no meu percurso, abriram portas e fundaram uma grande amizade. Obrigado.

Aos meus amigos, pelas saídas que ajudaram a espairecer da grande quantidade de trabalho. Obrigada pelo tempo e amizade.

Um especial abraço ao Filipe Freitas, amigo de infância e colega de curso com o qual travei longas horas de discussão, análise de problemas e produção de trabalhos para múltiplas disciplinas, momentos que trazem saudades da grande equipa que formamos.

À minha namorada Tânia Ferreira, deixo um beijo apaixonado pela compreensão e atenção pessoal dada durante os rotineiros domingos à tarde no sofá, que aproveitamos para namorar no intervalo dos estudos, vendo a sessão de cinema da televisão.

À minha família. Palavras não chegam para vos dizer o quão grato vos estou. Aos meus pais, pelo carinho, estímulo, esforço e compreensão. À minha irmã pelas longas conversas, brincadeiras e apoio durante os estudos. Pelos momentos que sorriram comigo porque venci, e pelos momentos em que me apoiaram quando pensei que não conseguiria. A vós agradeço a minha educação, maneira de ver o mundo e acima de tudo o vosso amor.
Obrigada mãe. Obrigada pai.

palavras-chave

Arquitectura de Computadores, Sistemas Reconfiguravéis, FPGAs, VHDL, Processadores MIPS, DETIUA-S3, Assembly, Microsoft .NET, Microsoft C#, iCmips 1.0, Xilinx ISE 9.2i

resumo

A influência da tecnologia digital é extremamente significativa em campos tão diversos como, por exemplo, o das telecomunicações onde as comunicações digitais adquiriram já uma individualidade própria, ou o do controlo onde os processadores digitais ocupam hoje um lugar indispensável.

A rápida evolução sentida na última década nas técnicas digitais, nomeadamente no domínio dos mini e micro-computadores, impõe uma constante reestruturação curricular que permita ir além do âmbito limitado das disciplinas de "Arquitectura de Computadores", "Sistemas Digitais", "Computação Reconfigurável" ou "Modelação e Síntese de Computadores", disciplinas estas leccionadas na Universidade de Aveiro aos cursos de Mestrado Integrado de Engenharia em Computadores e Telemática (MIECT) e Mestrado Integrado em Engenharia de Electrónica e Telecomunicações (MIEET).

O conhecimento do funcionamento dos computadores é muito importante para permitir que os estudantes de engenharia compreendam alguns conceitos da área de processamento de informação, a adaptabilidade de diversos algoritmos por simples programação, assim como a modularidade inerente aos sistemas digitais.

Nesta dissertação é apresentado um método que irá fazer uso dos conhecimentos adestrados nas disciplinas anteriormente mencionadas, aplicando-os de modo a que a aprendizagem em circuitos digitais, computação reconfigurável e arquitectura de computadores, vá para além do uso de software para simulação de um processador: a implementação real em hardware de um processador de arquitectura MIPS utilizando VHDL.

As ferramentas desenvolvidas nesta dissertação possibilitam aos alunos projectar, implementar e executar um processador MIPS Single Cycle de 32 bits na placa DETIUA-S3, que contém como componente central a FPGA Spartan3 XC3S400 da Xilinx.

Este documento apresenta as várias etapas da evolução das ferramentas criadas:

- A implementação de um processador MIPS Single Cycle utilizando a linguagem de descrição de hardware VHDL;
- A evolução de um protocolo de comunicação existente para interacção entre a placa DETIUA-S3 e um computador via USB *(Universal Serial Bus)*, tendo sido efectuadas alterações de modo a que permita o total controlo por software do processador desenvolvido;
- Uma aplicação, nomeada de "iCmips 1.0", que faz uso do protocolo de comunicação implementado possibilitando:
    - O controlo total do processador MIPS desenvolvido, desde efectuar a execução de ciclos de relógio, reset, leitura e escrita de valores no processador;
    - Configuração da aplicação mediante a arquitectura do processador implementado, como por exemplo, a dimensão das memórias, número de CPIs (*Cycles Per Instruction*) necessários, nome, dimensão e número de sinais ligados ao protocolo de comunicação, entre outros;
    - Configuração do conjunto de instruções MIPS assembly que o processador suporta e um assembler interno capaz de interpretar essas instruções (que não sejam do tipo vírgula flutuante);
    - Um editor de texto embutido que facilita o carregamento e escrita de programas assembly, sua verificação semântica e sintáctica e conversão em código máquina para posterior envio para as memórias de instruções e/ou dados do processador implementado.

O principal objectivo desta dissertação é então produzir uma nova ferramenta para ensino que possibilite aos alunos a aprendizagem dos mecanismos envolvidos em computação reconfigurável e arquitectura de computadores de uma forma simples, interactiva e cativante.

Os resultados obtidos demonstram a viabilidade e funcionalidade do sistema implementado, mas só com o envolvimento dos alunos na realização de exercícios como o desenvolvimento de alguns dos componentes do processador MIPS Single Cycle de 32 bits, inicialmente ocultados pelo professor, ou mesmo o desenvolvimento completo de novos processadores de arquitectura MIPS fazendo uso das ferramentas criadas para comunicação e depuração, se poderá avaliar as potencialidades e carências do projecto desenvolvido.

keywords

Computer Architecture, Reconfigurable Digital Systems, FPGA, VHDL, MIPS Processors, DETIUA-S3, Assembly, Microsoft .NET,
Microsoft C#, iCmips 1.0, Xilinx ISE 9.2i

abstract

The influence of digital technology is extremely significant in fields as diverse as, for example, the telecommunications where digital communications have already gained own individuality, or the control where the digital processors occupy an indispensable place today.

The rapid changes experienced in the last decade in digital techniques, particularly in the field of mini and micro-computers, requires a constant curricula restructuring that allows to go beyond the limited scope of the subjects of "Computer Architecture", "Reconfigurable Digital Systems" or "Processor Synthesis and Modeling", disciplines taught at the University of Aveiro within Computer Engineering curriculum and Electrical Engineering curriculum.

The knowledge of computer operations is very important to enable engineering students to understand some concepts of information processing, programming and modularity inherent to digital systems.

This dissertation presents a method that will deepen the expertise within the mentioned disciplines, so that the learning goes beyond the use of software for simulation of a processor to the actual implementation in hardware of a MIPS processor architecture using VHDL.

The tools developed in this dissertation enable the students to design, implement and execute a MIPS Single Cycle 32-bit processor on the DETIUA-S3 board, which contains as a central component the Xilinx Spartan3 XC3S400 FPGA.

This document presents the various stages in the development of tools created:

- Implementation of MIPS Single Cycle 32-bit processor using VHDL hardware description language;
- Evolution of an existing communication protocol for interaction between the DETIUA-S3 board and a host computer via USB (Universal Serial Bus), which suffered modifications to be able to control the developed processor by software;
- An application named "iCmips 1.0", which uses the communication protocol and allows the following:
    - Total control of the developed processor: generation of clock cycles, reset cycles, reading and writing values to the processor;
    - Configuration of software parameters, such as memory size, number of CPI (cycles per instruction), name, size, and number of signals used in communication protocol, etc., according to the implemented processor architecture;
    - Configuration of instructions supported by MIPS processor and used by an internal assembler;
    - A text editor, which facilitates writing assembly language programs, provides for syntactic and semantic code verification, and generates machine code that is further loaded to instruction/data memories of the implemented processor.

The main aim of this dissertation is to produce a new tool for education, enabling students to learn the mechanisms involved in “Reconfigurable Digital Systems” and “Computer Architecture” in a simple, interactive and engaging way.

The results obtained demonstrate the feasibility and functionality of the implemented system but only with the involvement of students (in conducting exercises such as the development of some components of the MIPS Single Cycle 32-bit processor, initially omitted by the teacher, or even the complete task of developing new MIPS processors architectures using the tools created for communication and debugging), it will be possible to evaluate the strengths and weaknesses of the developed project.

Índice

Lista de figuras

Lista de tabelas

Acrónimos

ALU *Arithmetic-Logical Unit*

ARM *Advanced RISC Machine*

AVR *Modified Harvard architecture 8 bits RISC*

CLB *Configurable Logic Block*

CLB *Configurable Logical Blocks*

CMOS *Complementary Metal-Oxide-Silicon*

CPLD *Complex Programmable Logic Device*

DARPA *Defense Advanced Research Projects Agency*

DCM *Digital Clock Manager*

DOD *United States Department of Defense*

E/S *Entrada/Saída*

EEPROM *Electronic Erasable Programmable Read Only Memory*

EPLD *Erasable Programmable Logic Device*

EPROM *Erasable Programmable Read Only Memory*

FPGA *Field-Programmable Gate Array*

GND *Ground (tensão de referência)*

GUI *Graphical User Interface*

HDL *Hardware Description Language*

I/O *Input/Output*

IDE *Integrated Development Environment*

IEEE *Institute of Electrical and Electronics Engineers*

IF *Instruction Fetch*

IOB *Input Output Block*

JTAG *Joint Test Action Group*

LUT *LookUp Table*

MaDMax *MIPS Digital Media eXtension*

MIECT *Mestrado Integrado em Engenharia de Computadores e Telemática*

MIEET *Mestrado Integrado em Engenharia de Electrónica e Telecomunicações*

PAL *Programmable Array Logic*

PBM *Prototyping Board Manager*

PC *Program Counter*

PLA *Programmable Logic Array*

PLD *Programable Logic Device*

PROM *Programmable Read Only Memory*

RAM *Random Accesss Memory*

RISC *Reduced Instruction Set Computer*

SIMD *Single Instruction, Multiple Data*

SLL *Shift Left Logical*

SRAM *Static Random Access Memory*

SRL *Shift Register LUT*

SRL *Shift Right Logical*

TTL *Transistor Transistor Logic*

VHDL *VHSIC Hardware Description Language*

VHSIC *Very High Speed Integrated Circuits*

VPP *Voltagem de Programação*

Capítulo 1 - Introdução

A melhor maneira de aprender sobre computadores é projectar e construir um. Computadores têm-se desenvolvido e adquirido arquitecturas complexas e sofisticadas (tais como múltiplas unidades de processamento num único processador ou memórias *cache*) e construir um processador em hardware não é tão concretizável numa disciplina com um semestre de duração.

Os sistemas reconfiguráveis prometem vir a ser preponderantes em muitos dos sistemas computacionais do futuro. A utilização de sistemas reconfiguráveis tem vindo a ser cada vez maior, sendo por vezes a única forma de atingir com sucesso características como: alto-desempenho, flexibilidade, baixo consumo de energia, rápida prototipagem, entre outros.

Enquadramento

"Arquitectura de Computadores", "Sistemas Digitais", "Computação Reconfigurável" e "Modelação e Síntese de Processadores" são disciplinas bastante comuns e relevantes no ensino superior em cursos de Engenharia de Computadores e Engenharia Electrónica.

É frequentemente comum aprender arquitecturas de computadores fazendo uso de software através do qual se efectua a implementação completa ou parcial das diversas partes que constituem um processador de computador. No entanto esta aproximação possui uma limitação: enquanto os alunos podem completar e simular os sistemas projectados, eles não possuem a oportunidade de implementar fisicamente e correr as arquitecturas projectadas.

O desenvolvimento em VHDL de um processador com Arquitectura MIPS para ensino visa proporcionar aos alunos que ingressam na Universidade Aveiro nos cursos de Mestrado Integrado em Engenharia de Electrónica e Telecomunicações (MIEET) e Mestrado Integrado em Engenharia de Computadores e Telemática (MIECT), uma nova perspectiva e abordagem às disciplinas anteriormente mencionadas e leccionadas nesta academia a ambos cursos. No entanto, o principal âmbito do desenvolvimento desta ferramenta para ensino destina-se para a disciplina de Computação Reconfigurável, leccionada aos alunos de MIECT.

Objectivos

No ensino da organização e arquitectura de computadores, compreender arquitecturas actuais e adquirir capacidades para projectar novas arquitecturas são objectivos importantes para os estudantes de engenharia.

Pretende-se então que o material desenvolvido no âmbito desta dissertação permita aos alunos atingir os seguintes objectivos:

- Aprender concretamente a estrutura e comportamento interno de um processador
- Adquirir sensibilidade das operações e mecanismos de comunicação realizados na arquitectura de um processador
- Interiorizar a relação das operações necessárias e mecanismos de comunicação do sistema desenvolvido incluindo software e hardware

O primeiro objectivo é importante pois pretende-se que o aluno fique familiarizado com a estrutura básica de um processador de modo a compreender as diversas técnicas de implementação de arquitecturas de processadores. A importância do segundo objectivo deve-se ao facto de possibilitar ao aluno adquirir o poder de decisão de optar pela melhor implementação de uma arquitectura face às condicionantes apresentadas. Por último, mas não menos importante, com o terceiro objectivo pretende-se que o aluno seja capaz de compreender de como o desempenho de um processador é afectado por cada um dos seus componentes.

Assim, o processador a ser projectado deverá ser capaz de dar apoio no ensino de sistemas reconfiguráveis, aprendizagem de uma linguagem de descrição de hardware e de arquitectura de computadores. Desta forma fica definido à partida que o processador reúna as seguintes características:

- O processador será projectado usando a linguagem de descrição de hardware (VHDL) sendo possível a implementação do mesmo numa FPGA da família SPARTAN 3 da Xilinx
- Deverá ser efectuada a descrição em VHDL estrutural, simulação e implementação do processador que realiza um subconjunto de instruções da arquitectura MIPS;
- Deverá ser constituído por componentes de forma a possibilitar a remoção de alguns e integração de outros desenvolvidos pelos alunos;
- Deverá ser elaborada uma ferramenta de depuração do processador implementado, de modo a que seja facilmente analisado e avaliado o comportamento e funcionamento do processador projectado e elaborado pelos alunos;

- A ferramenta de software deverá permitir a verificação semântica e sintáctica de programas escritos pelo utilizador na linguagem assembly simplificada e permitir a conversão dos programas no código máquina para uso no processador.

Estrutura da dissertação

Esta dissertação apresenta a descrição em VHDL de um processador com arquitectura MIPS, o conjunto reduzido de instruções MIPS de 32 bits, a sua implementação numa FPGA (Field Programmable Gate Array), a implementação de um protocolo de comunicação com o processador e uma ferramenta de software que faz uso do protocolo desenvolvido para interacção com o processador.

Esta dissertação encontra-se estruturada de forma que o leitor obtenha uma percepção progressiva sobre do trabalho elaborado, nomeadamente sobre a implementação das várias partes que o constituem.

Assim desta forma será apresentado no capítulo 2 uma breve introdução às tecnologias usadas de forma a dar a conhecer ao leitor alguns conhecimentos comuns para restante leitura deste documento seguindo-se a apresentação de alguns trabalhos realizados no mesmo âmbito do tema desta dissertação.

O capítulo 3 consistirá na abordagem tomada para realização dos objectivos propostos. Este capítulo é constituído por três subcapítulos referentes ao que se pode considerar as sucessivas e planeadas etapas para concretização dos objectivos: a elaboração do processador, o protocolo de comunicação implementado e a ferramenta de software desenvolvida. No final de cada subcapítulo é feita uma pequena análise do trabalho produzido.

Segue-se no capítulo 4, os resultados dos testes e análises efectuadas aos vários componentes produzidos. O facto de apresentar os testes elaborados e executados em cada uma das fases de desenvolvimento num capítulo separado é propositado e possui o intuito de dar a conhecer ao leitor todas as partes do sistema projectado sem que a leitura deste documento se torne cansativa ou confusa.

Por último, no capítulo 5 encontram-se redigidas as últimas reflexões sobre o trabalho futuro e conclusões sobre o material didáctico desenvolvido no âmbito desta dissertação.

Capítulo 2 - Estado da arte

Tecnologias usadas

MIPS

A arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages) teve nascimento na Universidade de Stanford nos Estados Unidos da América aproximadamente em 1981 quando o professor John Hennessy, lançou o desafio aos alunos de desenvolver uma arquitectura para um processador. A partir desta iniciativa surgiu o primeiro processador MIPS RISC (Reduced Instruction Set Computer). A ideia de obter uma arquitectura RISC não era inovadora pois já antes existiam projectos e esquemas para uma possível arquitectura deste tipo.

A filosofia RISC teve inícios nos anos setenta com a IBM e desde então foi desenvolvida e documentada em universidades e companhias de microprocessadores. O intuito desta filosofia é de optimizar e obter o máximo de desempenho de um processador e ao mesmo tempo simplificar o hardware de modo a conseguir custos razoáveis de produção [1]. Na arquitectura RISC o conjunto de instruções é baseado na abordagem *load/store*. Apenas o conjunto de instruções *load* e *store* permite o acesso à memória. Nenhuma das outras operações aritméticas ou de I/O interage com a memória. A simplificação neste tipo de arquitectura resulta numa rápida descodificação de instruções e facilidade de implementação.

Entre outras inovações como a criação de um conjunto próprio de instruções de chamadas ao sistema para operações comuns e repetitivas, a grande inovação resultante de uma longa discussão de ideias na equipa liderada pelo professor Hennessy, levou ao princípio base do desenvolvimento para a arquitectura MIPS: deve ser o compilador a ter maior responsabilidade de descer ao nível do hardware do que o oposto (o aumento do hardware para chegar ao nível do software). Tendo esta noção tornou-se evidente a necessidade de aumentar o desempenho de um processador sendo então necessário recorrer a uma técnica já na altura conhecida mas difícil de implementar: o uso de pipelines. Uma arquitectura *pipelined* não é mais que conseguir separar os múltiplos processos de execução de uma instrução em processos independentes, de modo a que se consiga ao mesmo tempo processar múltiplas instruções mas cada uma em fases diferentes de execução (semelhante a uma linha de montagem numa fábrica, ver Figura 1).

As primeiras arquitecturas MIPS desenvolvidas possuíam 32 bits (os registos e barramentos de 32 bits) no entanto posteriormente foram lançadas versões de 64 bits. Já existem múltiplas revisões do conjunto de instruções MIPS, incluindo MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, sendo o MIPS32 e MIPS64 (com implementações de 32 e 64bit respectivamente) as versões actuais. Existem ainda variadas extensões/vertentes desta arquitectura: o

MIPS-3D que consiste num simples conjunto de instruções de vírgula flutuante (floating-point SIMD -**S**ingle **I**nstruction, **M**ultiple **D**ata) dedicadas a tarefas comuns de renderização 3D; MIPS MDMX (MaDMax - MIPS Digital Media eXtension) que possui um conjunto mais extenso de instruções SIMD com inteiros usufruindo de registos de vírgula flutuante de 64bit; MIPS16e que é uma versão compacta de 16 bits de modo a que os programas sejam mais pequenos (com um intuito de dar resposta à codificação Thumb em arquitecturas ARM (Advanced RISC Machine); e uma das mais recentes adições à família MIPS o MIPS MT, com capacidade Multithreading muito semelhante ao sistema HyperThreading nos processadores Pentium 4 da Intel.

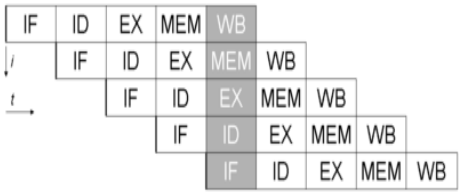


Figura 1 – As 5 fases da pipeline de uma arquitectura RISC

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). O eixo vertical representa as sucessivas (*i*) instruções e o eixo horizontal o tempo (*t*). Na coluna a escuro pode-se verificar que a primeira instrução encontra-se a terminar (em WB) e a mais recente e última instrução teve início (IF).

Devido à sua robustez de desempenho, formatos simples de instruções, diversidade e suporte de produtos, o processador MIPS, é neste momento um dos líderes de mercado. De facto podemos encontrar microprocessadores com tecnologia MIPS em gravadores de DVD, fotocopiadoras domésticas ou comerciais, modems, routers (por exemplo da companhia CISCO), consolas de jogos (como a Nintendo 64 e a Sony PlayStation 1, 2 e portátil), assim como em sistemas de TV como Series2 Tivo ou dispositivos móveis com o Windows CE [2,3].

A capacidade de projecto e implementação de sistemas computacionais implica o exercício prático. Só as lições teóricas de como uma arquitectura de um sistema funciona não chegam para que se adquira uma maturidade sobre a avaliação do problema proposto, os objectivos pretendidos a atingir e a decisão de qual caminho a tomar para a criação de uma arquitectura que processe e produza uma determinada informação.

A arquitectura MIPS é frequentemente leccionada em universidades pelo facto que permite aos docentes a escolha do nível de envolvimento a efectuar com os seus alunos face às restrições de tempo de ensino. Assim, a arquitectura MIPS pode ser apresentada aos alunos desde uma arquitectura simplificada que permita transmitir noções básicas de como um processador é implementado, a uma arquitectura mais complexa que permite a demonstração de como atingir melhores desempenhos e maior capacidades de processamento com arquitecturas de processadores com *pipeline*, co-processadores de virgula flutuante, *caching* ou *lock-ahead*.

VHDL

VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) é uma linguagem de descrição de sistemas electrónicos digitais.

A VHDL foi originalmente desenvolvida sob o comando do Departamento de Defesa dos Estados Unidos (*DARPA - Defense Advanced Research Projects Agency uma secção do DOD - United States Department of Defense*), em meados da década de 80, para documentar o comportamento de ASICs (application-specific integrated circuit) que compunham os equipamentos vendidos às Forças Armadas americanas. Isto quer dizer que a linguagem VHDL foi desenvolvida para substituir os complexos manuais que descreviam o funcionamento dos ASICs que até aquele momento, a única metodologia largamente utilizada no projecto de circuitos era a criação através de diagramas esquemáticos. O problema com esta metodologia é o facto de que desenho tem menor portabilidade, são mais complexos para compreensão e são extremamente dependentes da ferramenta utilizada para produzi-los.

Uma vez que o projecto VHSIC (***V**ery-**H**igh-**S**peed **I**ntegrated **C**ircuit*) era de alta prioridade militar e havia dezenas de fornecedores envolvidos, o departamento de defesa estava preocupado principalmente com as questões de portabilidade, documentação e compreensibilidade dos projectos. Cada um destes fornecedores actuava desenvolvendo partes dos projectos ou mesmo fornecendo componentes que viriam a se encaixar em outros sistemas maiores. Desta forma o departamento de defesa optou por desenvolver uma linguagem que servisse como

Tabela de Verdade de um Mux 2:1

Sel	a	b	Z
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

Equação Booleana:

$$Z = \overline{sel}.a + sel.b$$

VHDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity mux2_1 is
7     port(a,b,sel:in bit; z:out bit);
8  end mux2_1;
9
10 architecture behavioral of mux2_1 is
11    int1, int2, int3 : bit;
12 begin
13    z <= (not sel and a)or(sel and b);
14 end behavioral;
```

Figura 2 – Implementação de multiplexer 2:1 em VHDL (Comportamental)

Partindo da tabela de verdade obtêm-se intuitivamente a equação booleana de um multiplexer de 2:1. Muito facilmente se traduz a equação booleana em VHDL. O Exemplo de código VHDL descreve de forma comportamental a implementação de um multiplexer de 2 para 1bit.

base para troca de informações sobre estes componentes e projectos. Uma linguagem que, independente do formato original do circuito, pudesse servir como uma descrição e documentação eficientes do circuito, possibilitando os mais diferentes fornecedores e participantes a entender o funcionamento das outras partes, padronizando a comunicação (ver exemplo em Figura 2 e Figura 3) [4,5].

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity mux2_1 is
7     port(a,b,sel:in bit; z:out bit);
8  end mux2_1;
9
10 architecture structure of mux2_1 is
11    int1, int2, int3 : bit;
12 begin
13    n1:neg   port map(sel,int1);
14    a1:and2p port map(int1,a,int2);
15    a2:and2p port map(sel,b,int3);
16    o1:or2p  port map(int2,int3,z);
17 end structure;
```

Figura 3 – Implementação de multiplexer 2:1 em VHDL (Estrutural)

Código VHDL para implementar o mesmo Multiplexer que da figura anterior, mas usando a descrição estrutural.

O desenvolvimento da VHDL serviu inicialmente aos propósitos de documentação do projecto VHSIC. Entretanto, nesta época procurava-se uma linguagem que facilitasse o projecto de um circuito electrónico, ou seja, a partir de uma descrição textual, um algoritmo, desenvolver o circuito. A VHDL presta-se adequadamente a tais propósitos, podendo ser utilizada para as tarefas de documentação, descrição, verificação formal, síntese, simulação e teste.

Após o sucesso inicial do uso da VHDL, esta linguagem foi dada a conhecer publicamente, o que levou a ser padronizada pelo IEEE (*Institute of Electrical and Electronic Engineers*) em 1987. O facto de ser padronizada e de domínio público ampliou ainda mais a sua utilização e novas alterações foram propostas, como é natural num processo de aprimoramento. A linguagem sofreu uma revisão e um novo padrão mais actualizado que foi lançado em 1993. Em Junho de 2006 a comissão técnica de Accellera (comissão delegada pelo IEEE para trabalhar nas novas gerações e actualizações do standard) aprovou a nova versão conhecida como Draft 3.0 de VHDL-2006. Esta nova versão manteve a compatibilidade com as anteriores, surgindo novas facilidades da escrita e manipulação do código VHDL. Recentemente, em Fevereiro de 2008 foi lançada a mais actual versão da linguagem VHDL 4.0, conhecida informalmente como VHDL 2008 que veio a corrigir mais de 90 problemas encontrados durante o período de testes da versão 3.0 [5,6].

VHDL é uma linguagem muito versátil para qual existem múltiplos simuladores. É possível em VHDL escrever programas de teste (*testbench*) que verifiquem a funcionalidade do sistema projectado. São definidos estímulos para o sistema, de modo a simular a execução do sistema real podendo depois comparar os resultados obtidos com os esperados [7,8].

Estas características são semelhantes às da linguagem Verilog, também usada para os mesmos fins. VHDL é uma linguagem *strongly typed* (uma ou mais restrições de como é que operações com tipos de dados diferentes se podem relacionar) e como resultado é considerada por alguns como sendo superior à linguagem Verilog, no entanto é um tópico de intensa discussão entre os projectistas de sistemas já algum tempo. Ambas linguagens tornam relativamente fácil a projectistas inexperientes de sistemas produzir código que simula correctamente mas que na realidade não é sintetizável para aplicar num dispositivo real ou que torna-se demasiado grande e não praticável.

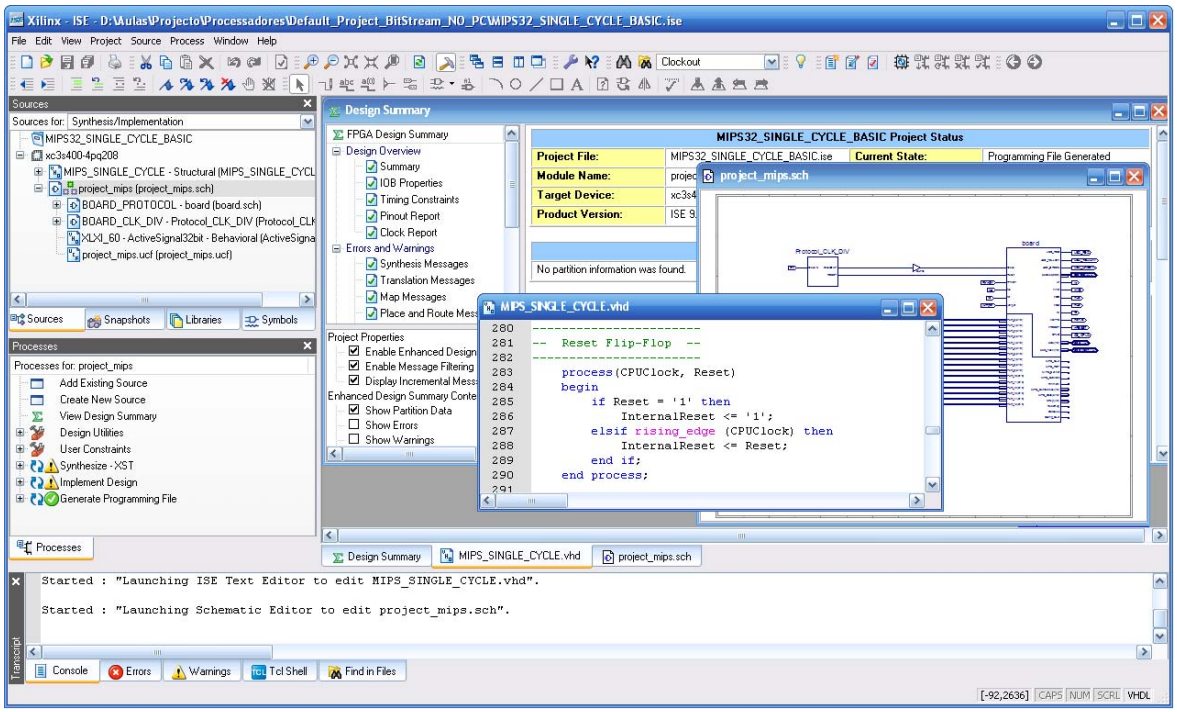


Figura 4 – Ambiente IDE (Integrated Development Environment) da ferramenta ISE Xilinx

VHDL não é uma linguagem do tipo "case sensitive". Pode-se projectar hardware em VHDL no ambiente IDE (como Xilinx, Figura 4, ou Quartus) e produzir o esquemático RTL do circuito. O hardware descrito pode ser verificado através de software de simulação (como o ModelSim) o que permite ver a forma das ondas de entrada e saída após ter sido aplicado a testbench apropriada.

A vantagem principal de VHDL quando usado para projecto de sistemas é que permite descrever o comportamento que se pretende ter com o sistema (modelo) e verificá-lo (simular) antes que as ferramentas de síntese traduzam o sistema em hardware real (portas e conectores), sendo também possível e desejável a posteriori (ver Figura 5).

Outra vantagem desta linguagem é o facto de permitir a descrição de um sistema concorrente (variados blocos, cada um com os seus sub-blocos, funcionando ao mesmo tempo). VHDL é uma linguagem do tipo *Dataflow*, ao contrário de algumas linguagens computacionais como BASIC, C ou código

assembly que são executadas sequencialmente, uma instrução de cada vez [7].

Por fim, o VHDL produzido é traduzido para hardware podendo ser mapeado em dispositivos lógicos programáveis como CPLD (*Complex Programmable Logic Device*) ou FPGA (*Field-Programmable Gate Array*) ou outros, passando a existir fisicamente hardware configurado e a executar o pretendido em vez de código VHDL a ser executado/simulado [8].

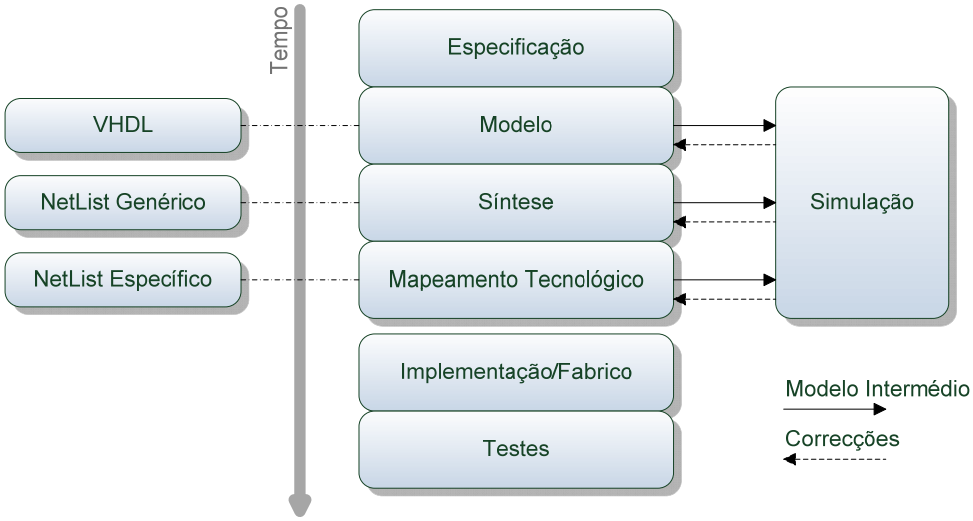


Figura 5 – Etapas de projecto em VHDL

A VHDL, bem como outras linguagens seguem um fluxo de projecto bem definido, composto de sete etapas: Especificação de Requisitos, Modelação, Síntese de Alto Nível, Mapeamento Tecnológico, Implementação e/ou Fabricação, Testes e Simulação. O tempo e o custo de cada etapa dentro de um projecto são bastante variáveis, dependendo da tecnologia utilizada para implementar o sistema.

FPGAs

A maior parte dos *chips* que encontramos hoje em dia, circuitos integrados que acompanham televisões, telemóveis, computadores, etc…, já vêm todos pré-programados, isto é, com as suas funcionalidades todas definidas no acto de fabricação. Existe no entanto outra categoria de circuitos integrados que possibilitam a sua programação. Nesta gama de dispositivos reconfiguráveis pode-se destacar a FPGA (*Field Programmable Gate Array*).

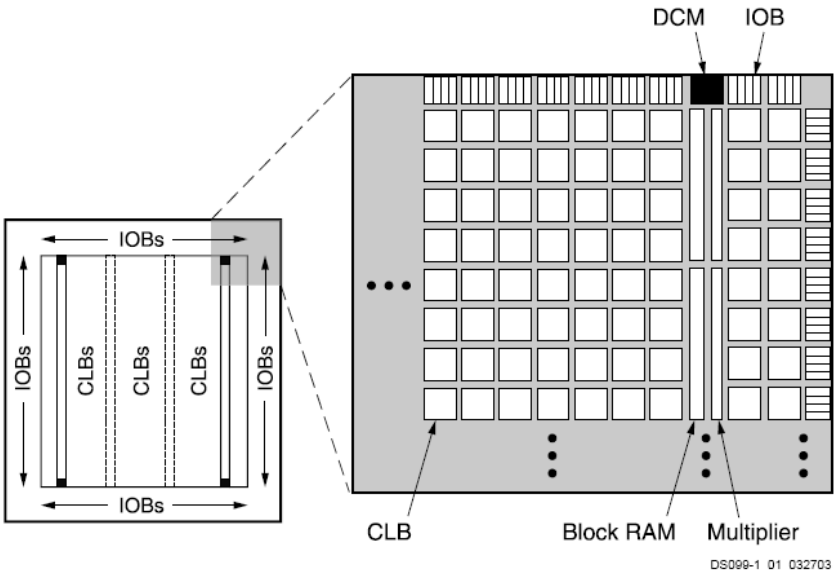


Figura 6 - Arquitectura da FPGA da família Spartan3
(imagem de [10])

A FPGA é um dispositivo semicondutor contendo componentes de lógica programável (Figura 6) chamados de blocos de entrada e saída (*IOB - Input/Output Block*), blocos lógicos configuráveis (*CLB - Configurable Logical Blocks*) e chaves de interconexão (*Switch Matrix*). Os blocos lógicos podem ser programados para efectuar operações de portas lógicas básicas como AND, OR ou funções combinatórias complexas. Em muitas das FPGAs, é comum encontrar blocos lógicos dedicados que incluem somadores, multiplicadores e elementos de memória.

As raízes históricas das FPGAs estão nos complexos dispositivos programáveis (*CPLDs – Complex Programmable Logic Device*) que tiveram origem nos inícios dos anos 80. CPLDs e FPGAs incluem um número significativo de elementos lógicos programáveis. Uma CPLD possui uma densidade de portas lógicas na ordem das centenas à ordem dos milhares enquanto FPGAs tipicamente possuem de uma ordem entre milhares a milhões (ver Tabela 1) [9].

A hierarquia de interconexões programáveis permite blocos lógicos estarem interligados à medida da necessidade do sistema desenvolvido, um tanto ao pouco análogo a ter um chip programável numa placa de testes (*breadboard*). Os blocos lógicos e inter-conectores podem ser configurados pelo utilizador ou projectista de sistemas, após a fabricação da FPGA, para implementar qualquer função lógica ou circuito – daí o nome "*field programmable*".

Família	Nº Portas de Sistema(K)	Característica	Processo de fabricação (nm)
XC2000	1,2 a 1,8	-	>500
XC3000	1 a 6	Low-power	
XC4000E	3 a 25	Low-power	
XC4000XL/XLA	5 a 85	High-density	
XC4000XV	120 a 220	High-density	
XC5200	3 a 23	Low-power	500
SPARTAN/XL	2 a 40	Low-power	
VIRTEX	50 a 1000	High-density	
SPARTAN-2	1 a 15	Low cost	220
SPARTAN-3	50 a 5000	Low cost	
VIRTEXII	40 a 10000	High-density	130
VIRTEXII-PRO	Power-PC inside		
VIRTEX4	13 a 200	Low-power	90
VIRTEX4-FX	Power-PC inside		
VIRTEX5	30 a 330		65

Tabela 1 – Evolução tecnológica das FPGAs da Xilinx

FPGAs são habitualmente mais lentas que os circuitos integrados dedicados (ASIC), não podem manipular sistemas tão complexos ou exceder uma certa potência de consumo. Mas as suas vantagens são notórias, entre as quais, diminuem o tempo de um produto chegar ao mercado, a capacidade de reprogramação permitindo corrigir erros ou falhas, e o seu uso é sinónimo de custos reduzidos na fase de desenvolvimento para a indústria de engenharia. Os circuitos do sistema são desenvolvidos em FPGAs reprogramáveis e posteriormente migrados para as versões menos flexíveis que tanto ou quanto se assemelham a ASICs.

Para configurar uma FPGA ou CPLD é especificado pelo utilizador um diagrama do circuito ou fonte de código em HDL (*Hardware Description Language*) que dirá a lógica com que o chip irá funcionar. A forma HDL, é capaz de ser uma das mais fáceis de trabalhar quando manipulando grandes e complexas estruturas, porque é possível especificar, identificar e referir-se às mesmas por nome em vez de ter um desenho/diagrama de cada um dos pedaços do circuito.

Para converter de diagramas/HDL para configurações físicas na FPGA ou CPLD, é necessário passar as fontes de código por software aprovado pelo fabricante, em que após diversas iterações é produzido um único ficheiro binário. Este ficheiro é por sua vez transferido para a FPGA/CPLD via a interface de série JTAG (*Joint Test Action Group*) ou para memória externa como uma EEPROM.

Uma das abordagens para optimizar a arquitectura tradicional das FPGAs foi combinar os blocos lógicos e interconexões com microprocessadores embutidos e periféricos. Exemplos deste tipo de tecnologia híbrida podem ser encontrados em produtos da Xilinx como as FPGAs Virtex-II PRO e Virtex-4, que possuem um ou mais processador PowerPC embebido na lógica da FPGA.

O Atmel FPSLIC é outro chip que usa um processador AVR (*Modified Harvard architecture 8 bits RISC*) combinado com a arquitectura de lógica programável da Atmel.

Uma alternativa ao uso de processadores hard-core é fazer uso de cores de processadores “soft” implementados no meio da lógica da FPGA.

Como referido anteriormente, muitas das FPGAs modernas possuem a capacidade de ser reprogramadas parcialmente e este é o conceito base de computação reconfigurável ou sistemas reconfiguráveis. O Mitrion Virtual Processor da Mitrionics é um CPU com base em FPGA, que se reconfigura a si próprio para conseguir produzir uma tarefa com sucesso. Este processador não suporta reconfiguração dinâmica durante a execução, mas adapta-se para um específico programa [12].

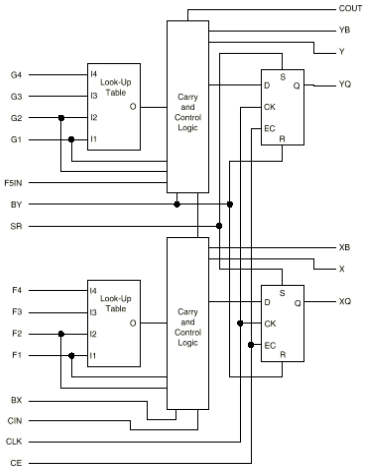


Figura 7 - Representação de uma Slice - unidade básica da família Spartan3
(imagem de [31])

Nas FPGAs da Xilinx a menor unidade lógica configurável e denominada *slice* de lógica e é mostrada na Figura 7. O *slice* é bastante versátil e pode ser configurado para operar como LookUp Tables (LUT) com quatro entradas e uma saída, RAMs distribuídas de 16 bits e registos de deslocamento de 16 bits. Na operação como LUT, os recursos adicionais como D-flip flops, multiplexadores, lógica de transporte (carry) dedicado, e portas lógicas, podem ser utilizados em conjunto com as LUTs para implementar funções booleanas, multiplicadores e somadores com palavras de comprimento bastante flexível. Na operação como SRL (Shift Register LUT), estes recursos adicionais podem ser utilizados para se implementar contadores, conversores série-paralelo e paralelo-série, entre outras funcionalidades. Os recursos adicionais mencionados podem ainda ser utilizados na interconexão de unidades lógicas para se implementar, por exemplo, multiplicadores, contadores, somadores e memórias com praticamente qualquer comprimento de palavra. O limite para este comprimento é determinado, entre outros, pela quantidade de unidades lógicas disponíveis na FPGA, que é proporcional à área do circuito integrado.

Além dos recursos padrões (*slices*) uma FPGA pode disponibilizar no arranjo bidimensional recursos bastantes sofisticados, tais como multiplicadores dedicados, MACs programáveis (multiplicador e acumulador, também denominados de slice XtremeDSP), blocos de memória, DCM (Digital Clock Manager utilizados para multiplicar ou dividir a frequência de um sinal de relógio), microcontroladores (IBM PowerPC) e trans-receptores de alto débito (que servem para implementar interfaces série para transferência de

bits a altas velocidades). A utilização de tais recursos embarcados possibilita optimizar o consumo de área (*slices*) e também desenvolver projectos mais sofisticados e eficientes [9-13].

Placa DETIUA-S3

Ao longo dos anos tem-se observado uma grande evolução a nível de placas de prototipagem, encontrando-se neste momento um mercado fértil no que se refere à oferta de placas de prototipagem. No entanto nem sempre é fácil encontrar uma placa que proporcione ou que abrange todos os interesses pretendidos entre os quais funcionalidade, aplicabilidade, flexibilidade e custos.

No ano lectivo 2004/2005 o Eng. Manuel Almeida juntamente com o professor Valeri Skliarov, desenvolveram a placa de prototipagem DETIUA-S3, que tem como componente central uma FPGA Spartan-3 da Xilinx (ver Figura 8) [14].

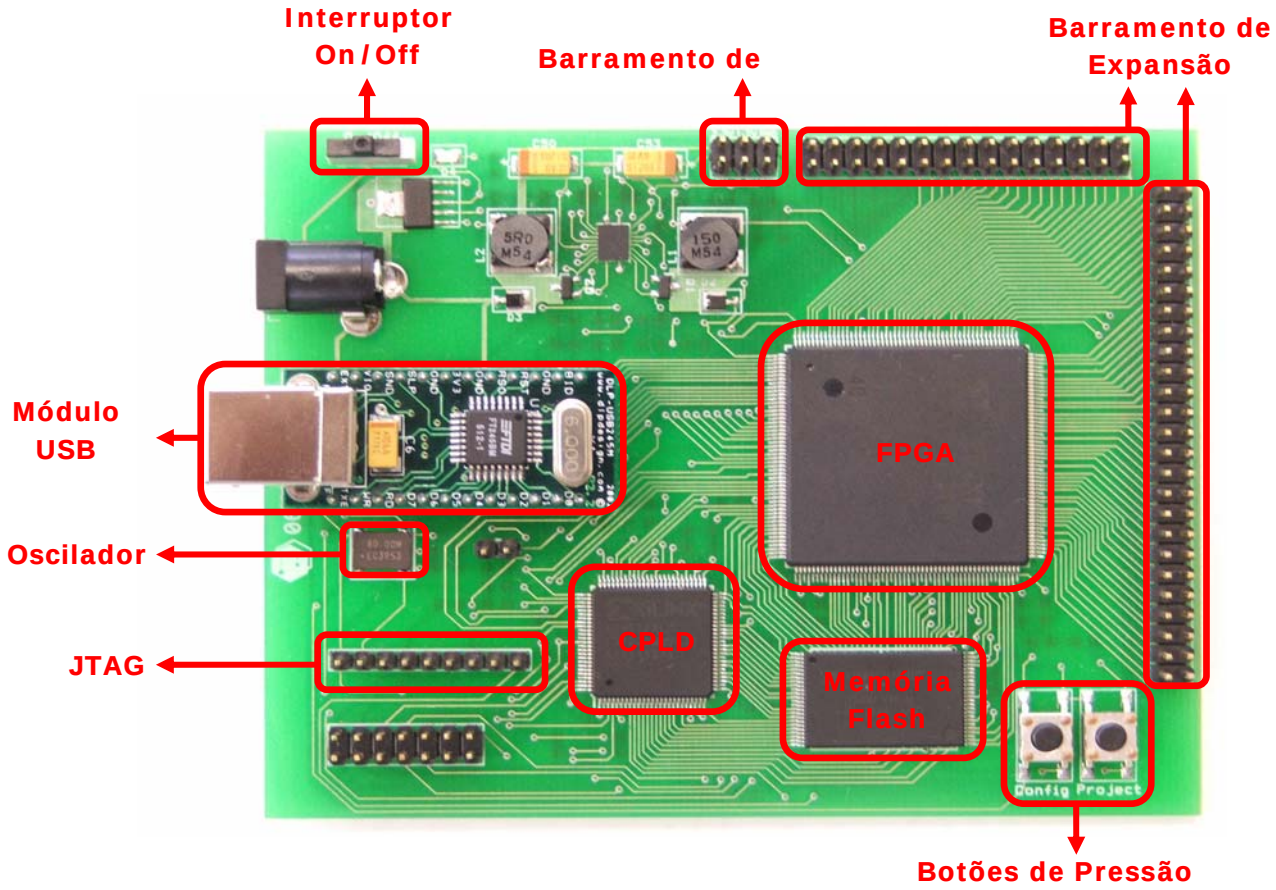


Figura 8 – Placa DETIUA-S3 e localização de alguns dos componentes na placa

Dos componentes que fazem parte da arquitectura da placa DETIUA-S3 (ver Figura 8 e 9), é relevante salientar:

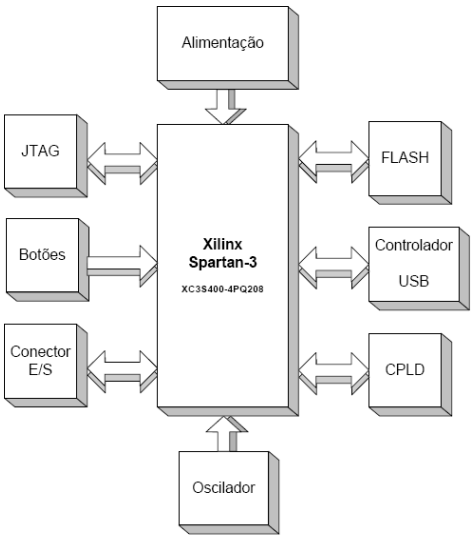


Figura 9 – Diagrama de blocos da placa DETIUA-S3
(imagem de [14])

- O componente central: a FPGA Spartan-3 da Xilinx, como anteriormente mencionado, que contém entre outros, 400 mil portas de sistema, 56Kb de RAM distribuída, 16 multiplicadores, 141 pinos de entrada e saída;
- O módulo de comunicação e também de alimentação: DLP-USB245M da DLP Design, tendo componente central o FT245BM da FTDI, consiste numa interface USB-paralelo (Universal Serial Bus - FIFO), compatível com USB 1.1 e USB 2.0. Ligando este módulo a um computador, é possível transferir dados a um máximo de 1 MB/s e, ao mesmo tempo, fazer uso da tensão de 5V fornecida pela porta USB à qual está ligada. Assim, um único cabo permite realizar toda a gestão e alimentação da placa a partir de um computador. Existe no entanto a possibilidade de substituir este módulo por uma interface Bluetooth em que todo o processo de comunicação é idêntico (possuindo obviamente menores velocidades de transmissão de dados), no entanto a alimentação da placa passará a ser externa;
- Uma memória flash que tem uma capacidade de 2 MB, com sectores de 64 Kb, exceptuando os últimos quatro. A memória encontra-se dividida em três zonas lógicas tal como ilustra a Figura 10;
- Uma CPLD XC9572XL da Xilinx: que possui a finalidade de auxiliar o processo de configuração da FPGA, utilizando o barramento de endereços e pinos de controlo da memória flash, lê o ficheiro de configuração (bitstream) para a FPGA;
- Um oscilador de 80MHz: este sinal de relógio é utilizado pela FPGA e CPLD num dos pinos de global clock;
- Dois barramentos de expansão: um com 50 pinos e outro com 30 pinos de entrada/saída, que permitem a ligação de placas de extensão, algumas já desenvolvidas para efeitos de conectividade a diversos dispositivos como teclado, rato, VGA, porta RS232, etc.;
- Barramento de alimentações: este barramento disponibiliza além da massa (GND), as tensões de 1.2V, 2.5V, 3.3V e 5V.

- Barramento JTAG: através do qual é possível efectuar a configuração da FPGA e CPLD. Este método de configuração, Boundary Scan mais conhecido como JTAG, permite uma configuração directa dos dispositivos;
- Dois botões de pressão: o botão Config permite ao utilizador configurar a placa com o ficheiro de configuração por omissão; e o botão Project permite ao utilizador disparar a configuração do ficheiro transferido que contém a informação sobre o projecto do utilizador.

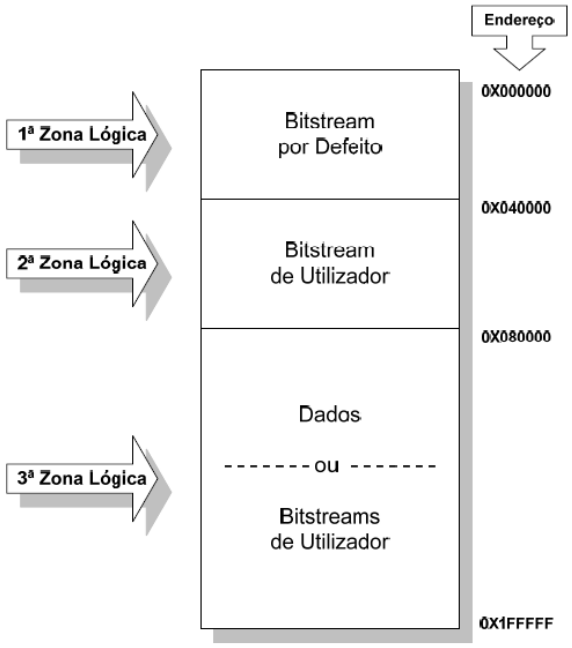


Figura 10 – Zonas lógicas da memória flash
(imagem de [14])

Diagrama ilustrativo de como a memória flash se encontra dividida e a que corresponde cada uma das zonas lógicas.

Uma interface *user-friendly* foi desenvolvida para efectuar a ponte entre o PC e a placa DETIUA-S3. O PBM (Prototyping Board Manager) [15] permite efectuar toda a interacção com a placa, deste modo, pode-se transmitir facilmente os bitstreams (informação de configuração da FPGA; constituída por um vector de bits denominado de BITSTREAM) gerados para placa efectuando a configuração da FPGA (mais informações mais à frente).

A DETIUA-S3 já contribuiu como suporte base para inúmeros projectos de investigação e aplicações didácticas entre alunos e professores da Universidade de Aveiro, sendo alguns já publicados (algumas referências podem ser consultadas em [16]).

PBM

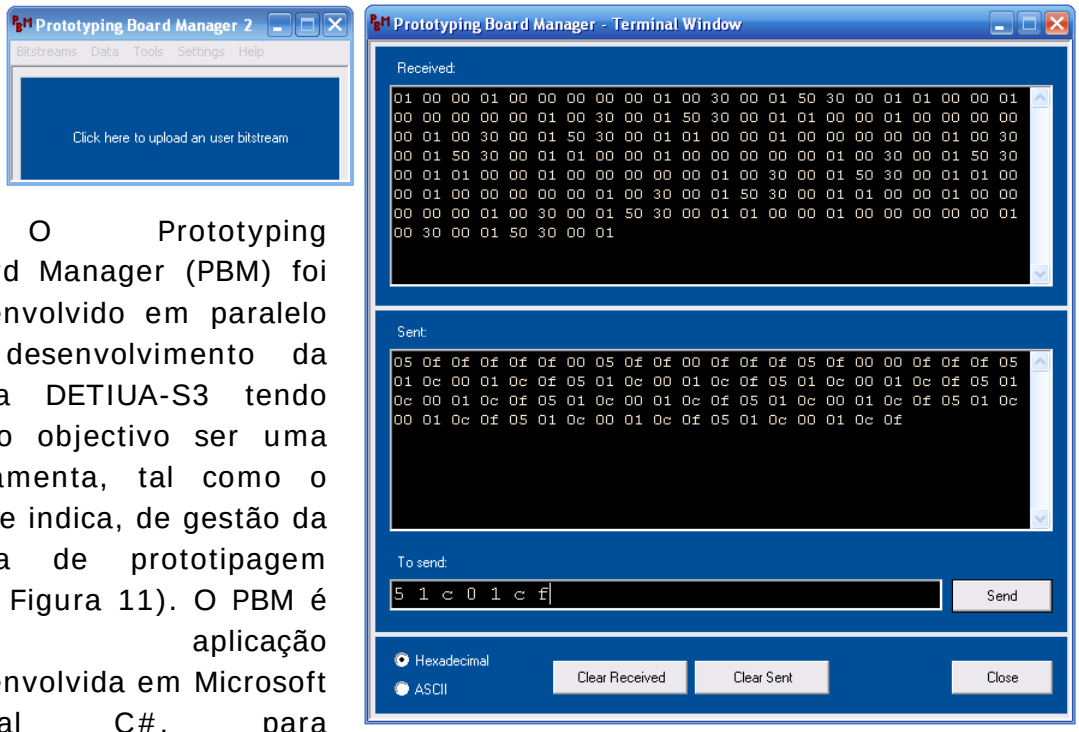


Figura 11 – Janela principal e consola de comunicação do PBM versão 2.0

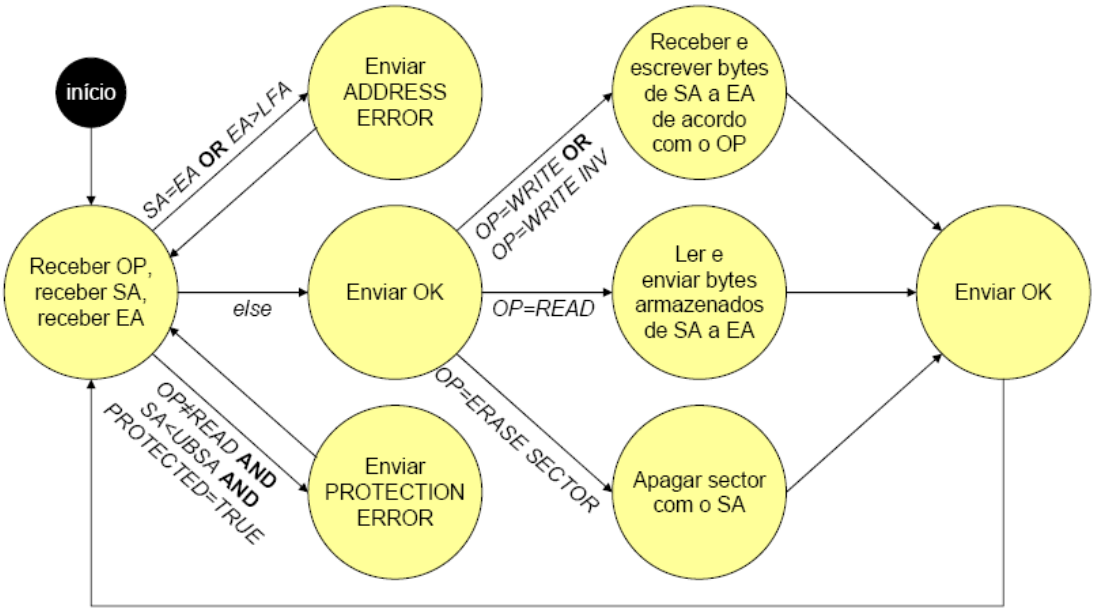
O Prototyping Board Manager (PBM) foi desenvolvido em paralelo ao desenvolvimento da placa DETIUA-S3 tendo como objectivo ser uma ferramenta, tal como o nome indica, de gestão da placa de prototipagem (ver Figura 11). O PBM é uma aplicação desenvolvida em Microsoft Visual C#, para plataforma Windows, que possui a capacidade de comunicar com o módulo USB (ou Bluetooth) da placa DETIUA-S3 fazendo uso da importação dos drivers FTD2XX, fornecidos pela empresa FTDI fabricante do chipset do módulo USB [15].

O PBM foi projectado para trabalhar sobre o protocolo de comunicação definido pela bitstream por omissão (*default bitstream*). Analisando a

Figura 12 representativa da máquina de estados implementada com a bitstream por omissão verifica-se que para efectuar uma única operação é necessário enviar o código da operação e caso seja preciso para a operação os endereços de inicio e fim de leitura/escrita da memória.

Através desta aplicação o utilizador tem a possibilidade de adicionar e configurar parâmetros da arquitectura e comunicação com a placa de prototipagem em uso, transferir múltiplas bistreams (configuração por omissão, de utilizador ou alternativas), ler ou carregar dados de/para a memória flash ou através do seu terminal interno comunicar directamente com o placa DETIUA-S3 utilizando os códigos implementados no protocolo de comunicação para identificação das operações a efectuar (Erase, Read, Write ou Write Inverse).

O PBM encontra-se preparado para lidar com erros provenientes de falhas de comunicação ou alguma anomalia verificada enquanto em comunicação com a placa de prototipagem.



Legenda:

OP (Operation)	Parâmetro de 1 byte, para especificar a operação de entre as seguintes: ERASE SECTOR, READ, WRITE e WRITE INV.
SA (Starting Address)	Parâmetro de 3 bytes para especificar o endereço da memória flash inicial para a operação solicitada; válido se estiver entre 0 e EA.
EA (Ending Address)	Parâmetro de 3 bytes para especificar o endereço da memória flash final para a operação solicitada; válido se estiver entre SA e LFA.
LFA (Last Flash Address)	Constante específica da placa, que indica o endereço da última posição da memória flash (endereçável com endereços de 3 bytes).
UBSA (User Bitstream Starting Address)	Constante específica da placa, que indica o endereço inicial da segunda zona lógica da memória flash (*user bitstream area*).
PROTECTED	Sinal de 1 bit, da placa, para indicar se o mecanismo de protecção da primeira zona lógica da memória flash (*default bitstream*) está activo ou não.
OK	Mensagem de controlo que indica a validação dos parâmetros do pedido de operação e sincroniza a comunicação no final; constante de 1 byte com o valor 0x01.
PROTECTION ERROR	Mensagem de controlo que nega um pedido de operação ao tentar escrever na primeira zona lógica da memória flash (*default bitstream area*) com o mecanismo de protecção activo; constante de 1 byte com o valor 0x02.
ADDRESS ERROR	Mensagem de controlo que nega um pedido de operação devido a um conjunto inválido de parâmetros SA e EA; constante de 1 byte com o valor 0x03.
ERASE SECTOR	Constante atribuível ao parâmetro OP para solicitar o apagar de um sector; constante de 1 byte com o valor 0x04.
READ	Constante atribuível ao parâmetro OP para solicitar uma leitura da memória flash; constante de 1 byte com o valor 0x05.
WRITE	Constante atribuível ao parâmetro OP para solicitar uma escrita simples na memória flash; constante de 1 byte com o valor 0x06.
WRITE INV (Write Inverted)	Constante atribuível ao parâmetro OP para solicitar uma escrita na memória flash com inversão da ordem dos bits de cada byte; constante de 1 byte com o valor 0x07.

Figura 12 – Máquina de estados da configuração por omissão
(imagem de [15])

Microsoft C#

Uma linguagem de programação é um método padronizado para expressar instruções para um computador. É um conjunto de regras sintácticas e semânticas usadas para definir um programa de computador. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai actuar, como estes dados serão armazenados ou transmitidos e quais acções devem ser tomadas sob várias circunstâncias.

A primeira linguagem de programação de alto nível amplamente usada foi Fortran, criada em 1954 e desde então as linguagens de programação foram sofrendo múltiplos desenvolvimentos, optimizações e melhoramentos. Esta evolução teve início em linguagens de baixo nível (Assembly) sendo esta considerada a primeira geração, seguida das primeiras linguagens fortemente utilizadas como Fortran e ALGOL (ALGOrithmic Language), linguagens declarativas e modulares (Pascal e C), linguagens de consulta (SQL) e por fim linguagens lógicas (Prolog, Caml e Mercury).

C# (lê-se C Sharp) é uma linguagem de programação orientada a objectos criada pela Microsoft em conjunto com a arquitectura Microsoft .NET (*dot NET*). Embora existam várias outras linguagens que suportam essa tecnologia (como VB.NET, C++, J#), C# é considerada a linguagem símbolo do .NET. Anders Heijlsberg, principal desenvolvedor e líder do projecto, que antes já haver estado envolvido em projectos como Visual J++, Borland Delphi, e Turbo Pascal, juntou uma pequena equipa com intuito de construir uma nova linguagem a qual na altura conhecida de “Cool”, posteriormente renomeada para “C#”. A título de curiosidade, na verdade o "#" de C# refere-se ao sinal musical (sustenido), que aumenta em 1/2 tom uma nota musical.

A Microsoft submeteu o C# a ECMA para uma padronização formal em Dezembro de 2001 e a ECMA liberou a especificação ECMA-334 da Linguagem C#. Em 2003 tornou-se um padrão ISO (ISO/IEC 23270). Esta etapa deu início a algumas implementações em desenvolvimento, destacando:

- Mono, implementação open source da Novell.
- dotGNU e Portable.NET da Free Software Foundation.
- BDS 2008 da CodeGear.

C# está de tal forma ligado à plataforma .NET que não existe o conceito de código não-gerenciado (unmanaged code) em C#. Suas estruturas de dados primitivas são objectos que correspondem a tipos em .NET. A desalocação automática de memória por *“garbage colletor”* além de várias de suas abstracções tais como classes, interfaces, delegados e excepções são nada mais que a exposição explícita dos recursos do ambiente .NET.

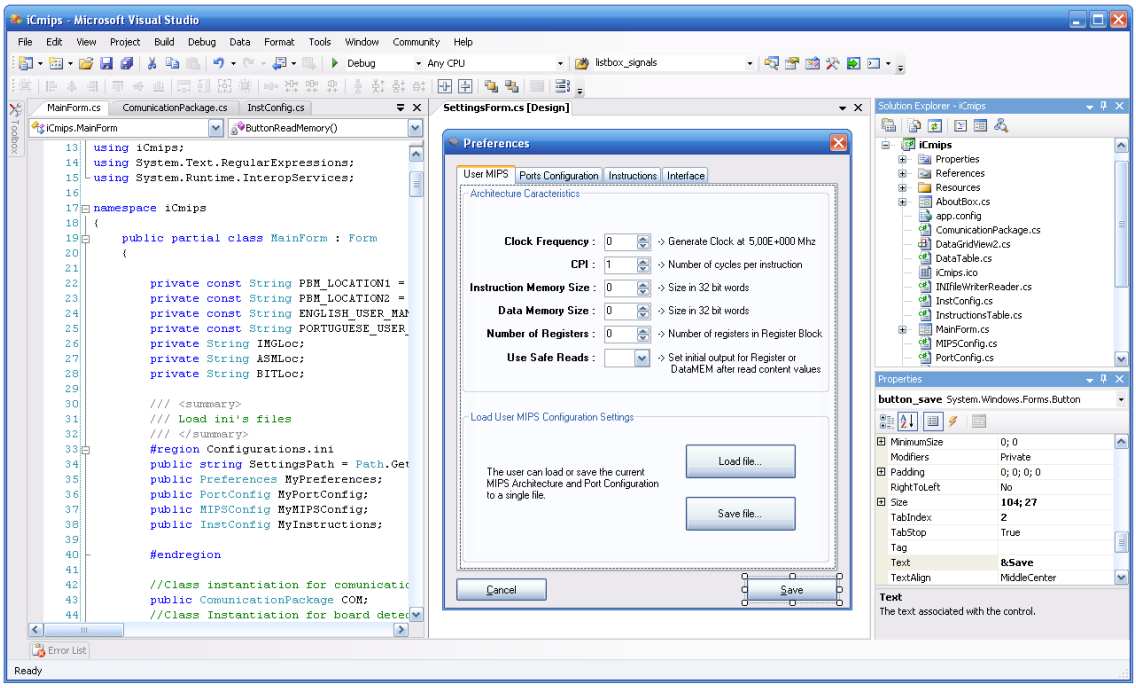


Figura 13 – Ambiente IDE do Microsoft Visual Studio 2005

Ambiente IDE usando a linguagem C#, na imagem à esquerda o editor de texto, no centro construtor visual de janelas, à direita em cima os ficheiros associados ao projecto e à direita em baixo as propriedades de um dos objectos visualmente representados.

Apesar de C# ser frequentemente comparado com Java, existem uma série de diferenças importantes em ambas as linguagens. Por exemplo, o Java não implementa propriedades, mas permite a utilização de métodos Get e Set que realizam o mesmo processo. Outros detalhes são:

- O Java não implementa o goto como estrutura de controlo, mas o C# sim, apesar de ser pouco usual.
- O Java utiliza comentários Javadoc e o C# utiliza comentários baseados em XML.
- O C# possui indexadores. O Java tem Listeners;
- O Java utiliza a JVM, o C# o .Net Framework, Mono e DotGnu.
- O Java pode ser compilado em qualquer plataforma, o C# possui compiladores para Windows (ver Figura 13), Windows Mobile, Linux, Mac OS X e Solaris. A plataforma .Net da Microsoft suporta compiladores para Windows e Windows Mobile (.Net Compact Framework), já a plataforma Mono permite compiladores para Windows, Linux, Mac OS X e Solaris.

No final, ambas possuem recursos similares, sendo possível o programador escolher a linguagem com a qual mais simpatiza [17-19].

Trabalhos existentes

O conceito de projectar um processador de arquitectura MIPS para ensino não é novo, de facto, já há muito que se desenvolvem projectos que procuram dar resposta a como obter a melhor implementação face aos conteúdos dados na academia para a qual o projecto é implementado. De seguida apresento alguns trabalhos desenvolvidos no mesmo âmbito desta dissertação segundo os quais procurou-se analisar a abordagem realizada ao problema, ferramentas utilizadas e mecanismos implementados que facilitem a aprendizagem.

No artigo publicado "*A reconfigurable microprocessor teaching tool*" [20], Diab *et al* deu a conhecer um pacote de software capaz de demonstrar graficamente aos alunos o comportamento interno de um processador de 8 bits. O pacote incluía um assembler, a unidade de controlo e o simulador gráfico. A presença do assembler permitia aos alunos escolher entre assembly ou código máquina para carregar e correr no processador. O simulador gráfico apresentava os vários processos e etapas de como as instruções eram tratadas nos vários componentes do processador. Usando esta ferramenta os alunos puderam observar de como as diferentes partes do processador interagiam, e como a combinação das suas funções recriam aquilo que é um processador.

Em 1997, Salcic e Smailagic [21] projectaram um microprocessador configurável ao qual deram o nome de SimP. Apesar o processador desenvolvido não ser vocacionado para o ensino, possui muitos atributos que fariam deste microprocessador uma ferramenta óptima para ensino, incluindo o suporte a um conjunto de instruções de 16 bits, registos, ALUs e um datapath simples. As diferenças entre o processador SimP e MIPS apresentado no livro Patterson and Hennessy [22] são notórias, de tal maneira que se fosse usado um processador SimP nos laboratórios mas adestrando o design do processador de Patterson and Hennesy na teoria, seria pouco produtivo. Outra opção era adoptar unicamente o SimP e toda a sua documentação mas mais uma vez os textos são demasiado dispersos para ajudar na aprendizagem de arquitectura de computadores ou sistemas reconfiguráveis [23].

Na tese de Mark Holland de 2002 [23], podemos encontrar informação de como foi efectuada a implementação de um processador single-cycle numa FPGA de modo a que o seu funcionamento e arquitectura assemelhasse à proposta que se encontra e é actualmente leccionada na Universidade de Aveiro, no livro de Patterson and Hennesy [22]. A placa utilizada para implementação foi a XESS-XSV que inclui uma FPGA Virtex XCV300 da Xilinx, dois blocos de 512K x 16 SRAM, uma porta paralela de interface e um botão de pressão. Todo o datapath da arquitectura do processador foi implementado

na FPGA à excepção das memórias de dados e instruções devido à falta de espaço de armazenamento na FPGA. A porta paralela foi usada para carregar configurações para a FPGA, assim como comunicar com o PC, o que é necessário para a ferramenta de depuração desenvolvida. O botão de reset foi usado meramente para efectuar o reset ao processador. A implementação contempla o controlo, fazendo uso de um relógio de alta frequência para comunicação entre os componentes na FPGA e as memórias SRAM, relógio este que se encontra oculto aos alunos. Um sinal de relógio de mais baixa frequência (16 vezes mais baixa) é utilizado para o processador (seu relógio e controlo) que é usado pelos alunos.

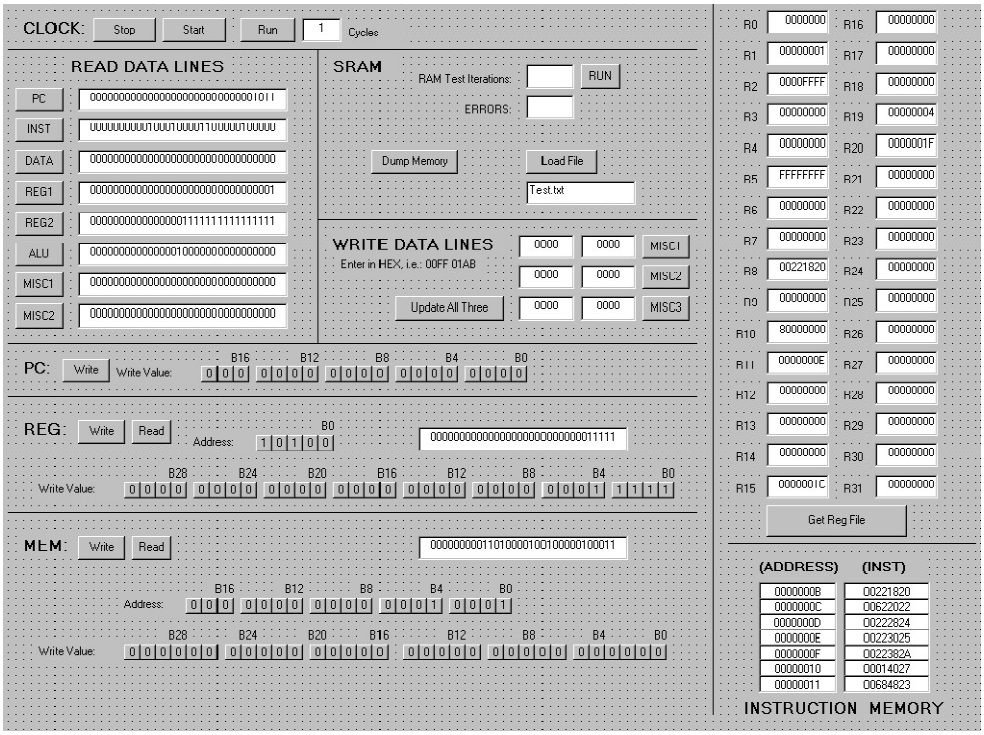


Figura 14 - Interface do depurador desenvolvido por Mark Holland
(imagem de [23])

O desenvolvimento de uma ferramenta de software (Figura 14) foi necessário para tornar possível a utilização do processador implementado aplicável numa sala de aula. A ferramenta permite aos alunos observar e total controlo dos estados internos do processador, o que é crítico para o projecto e depuração do mesmo. A ferramenta foi desenvolvida em Visual Basic fazendo uso da porta paralela para comunicação e possibilitando dois métodos de controlo do sinal de relógio: executar um número específico de ciclos ou executar ciclos de relógio até o utilizador decidir parar. O estudante pode então ler os dados seleccionando uma das fontes existentes: program counter, instruction memory, data memory, registers, ALU, etc (ver Figura 14). Os alunos podem ler ou escrever nas memórias ou registos necessitando para

isso apenas indicar a localização de leitura/escrita e no caso de escrita, que dados a escrever.

A implementação de um processador MIPS com capacidade de processar todo o conjunto de instruções existente foi logo à partida excluída pois envolveria ter de despender demasiado tempo das aulas e alunos, assim como recursos da FPGA. Assim apenas implementaram um conjunto de oito instruções: NOR, SUBU, LW, SW, BGEZ, JALR, SYSCALL and BREAK.

Foi ainda implementado um assembler que tem como objectivo o mapeamento de qualquer instrução MIPS que não seja de vírgula flutuante num conjunto de oito instruções. Este assembler foi escrito em LEX e YACC, verificadores léxicos e gramáticos utilizados em Linux. O resultado à saída do assembler é um ficheiro contendo o código máquina de um conjunto de instruções assembly dum ficheiro que é identificado à entrada. Este ficheiro de saída pode ser carregado directamente para o processador.

Neste projecto os alunos têm acesso à placa XSV, um computador com o software da Xilinx Foundation, o depurador do processador desenvolvido, o conector da porta paralela, um projecto contendo a framework base do processador e a fonte de alimentação da placa. Com estas ferramentas e algum auxílio de navegação pelas mesmas, em pouco tempo ficam aptos para desenvolverem partes ou novos processadores.

A implementação do processador MIPS single-cycle e Framework desenvolvida implicou a utilização de 22% das LUT e 25% das Block RAM da FPGA Xilinx Virtex XCV300 e a FPGA corre a uma frequência máxima de 25MHz o que se traduz num processador de 1.5MHz. A implementação do processador *pipelined* resultou na utilização de 29% das LUTs e 25% das BRAMs, à uma frequência de execução de 23 MHz (1.4MHz clock processor).

No ano de 2005 outro projecto semelhante foi desenvolvido por Diarmaid O'Cearuil [24] sendo seu trabalho a evolução de projectos anteriores já existentes [25]. O trabalho final consiste num processador em que a arquitectura é apresentada de forma esquemática, demonstrando as interligações dos vários blocos correspondentes aos componentes do processador.

Este projecto consistiu na evolução da arquitectura de 8 bits para 32 bits e num melhoramento a nível de funcionalidade do projecto para ensino. A arquitectura implementada possui um subconjunto de instruções suportadas as quais podem ser consultadas em [24]. De uma forma geral, o design do processador segue a estruturas descritas em Patterson and Hennessy [22].

Uma interface gráfica foi desenvolvida, em que o aluno/professor tem a possibilidade de seleccionar os componentes a ocultar (ver Figura 15). A ferramenta tem a capacidade, face às opções efectuadas, montar um projecto que um aluno pode abrir com a ferramenta da Project Navigator da Xilinx. Nesse momento o aluno pode desenvolver os módulos eliminados de forma a completar o processador e posteriormente verificar a sua coerência de

funcionamento. O aluno, após a verificação e análise das formas de ondas originadas pelos diversos componentes pode implementar o processador desenvolvido na placa Virtex II XC2V1000 da Xilinx. Neste projecto foram usados mecanismos de visualização da execução física do processador, utilizando o partido de as placas de prototipagem possuírem componentes embutidos (LEDs de 7 segmentos, LEDs simples e botões) ou permitirem a anexação de placas de extensão com os componentes para expressar valores de sinais do processador.

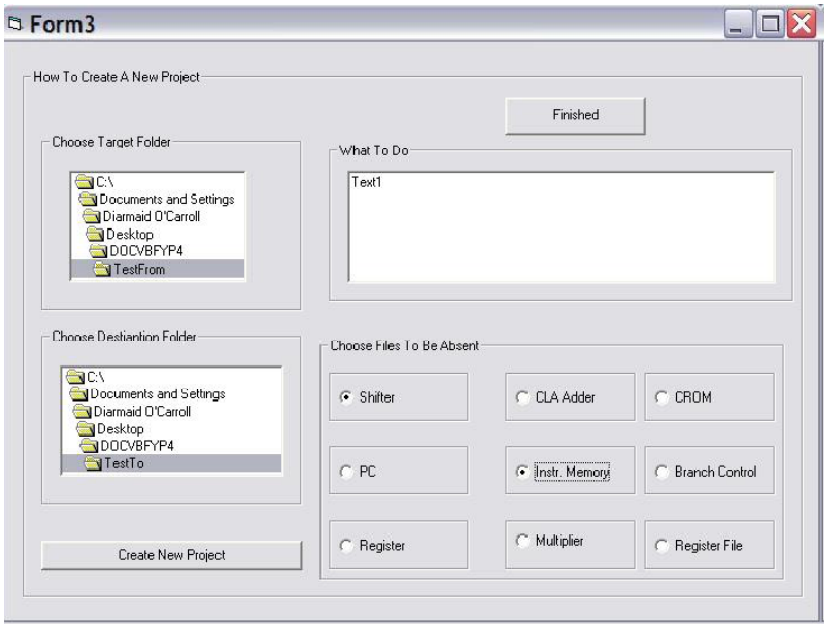


Figura 15 - Interface do GUI (Grafical User Interface) desenvolvido por Diarmaid O'Cearuil

Janela do momento de criação de um novo projecto (imagem de [24])

Muitos outros trabalhos e documentações foram realizados nesta área [26-29], uns mais simples e concretos, outros mais complexos e inúmeras vertentes, mas todos sempre com o mesmo intuito de proporcionar um mecanismo simples de compreensão, fácil de utilização e com uma componente didáctica de aprendizagem de sistemas reconfiguráveis, arquitecturas de computadores e implementação de processadores usando FPGAs.

Analisando os trabalhos apresentados, as abordagens tomadas por Diab *et al* [20], Salcic e Smailagic [21], e Mark Holland[23] proporcionaram atingir os mesmos objectivos de ensino de uma arquitectura de processador. No entanto quando analisadas comparativamente, verificam-se diferenças que de certa forma reflectem a evolução da tecnologia disponível no momento de desenvolvimento do projecto.

O facto de Salcic e Smailagic [21] desenvolverem um projecto que já envolvesse uma componente física, ao contrário de um ambiente puramente

de simulação como o de Diab *et al* [20], comprova o facto de que só o ensino da teoria e simulação da arquitectura não é o suficiente. É bastante mais cativante e produtivo a nível de ensino o facto de proporcionar aos alunos a oportunidade de desenvolverem hardware físico que implemente o estudado.

Na abordagem de Mark Holland [23] a possibilidade de projectar uma arquitectura MIPS, utilizar como apoio a literatura de Patterson e Hannessy [22], e ter ferramentas complementares que proporcionem a interactividade com o processador, dá inicio a uma nova perspectiva sobre como efectuar o ensino de sistemas reconfiguravéis, arquitecturas de processadores e depuração de resultados.

O projecto desenvolvido por Diarmaid O'Cearuil [25], apresenta a inovação de possuir uma ferramenta que possibilita ao aluno a criação de um projecto contendo apenas os módulos que pretende implementar, inovação que o projecto de Mark Holland [23] não possui e é de todo interessante para a sala de aula, mas em contra partida, o seu projecto encontrou vários problemas de sincronismo de relógios que não ficaram resolvidos, impossibilitando a visualização nos LEDs de 7 segmentos a execução do processador desenvolvido.

Visto isto, é importante que o projecto de uma ferramenta para ensino capte todas as vantagens observadas nos vários projectos apresentados, melhore a harmonia entre o que é leccionado nas aulas teóricas e o que é produzido nos laboratórios e possibilite a aprendizagem de uma forma faseada, progressiva e envolvente de todas as áreas científicas abrangidas: VHDL, sistemas reconfiguráveis, arquitectura de computadores e assembly.

Capítulo 3 - Descrição do trabalho realizado

Breve descrição explicativa à abordagem efectuada

Após a leitura de outros trabalhos realizados no mesmo âmbito deste e reflexão dos objectivos pretendidos para o processador a ser implementado surgiram diversificadas ideias de implementação.

Ficou então notória a necessidade de desenvolver os variados componentes e toda a estrutura de um processador de arquitectura MIPS utilizando VHDL; a utilização de um mecanismo ou ferramentas que permitam a comunicação com o processador para obter informações do estado, sinais e valores internos do mesmo; a criação e disponibilização de material didáctico para futura distribuição entre alunos e utilização nas aulas.

Do estudo dos resultados obtidos e erros cometidos nos trabalhos analisados procurou-se efectuar uma planificação ao projecto que fosse possível concretizar o seguinte plano de trabalhos:

- Descrição em VHDL e simulação de componentes principais do processador.
- Descrição em VHDL estrutural, simulação e implementação do processador que realiza um subconjunto de instruções da arquitectura MIPS.
- Desenvolvimento da ferramenta de software que permita:
    - efectuar a verificação sintáctica de programas escritos pelo utilizador na linguagem assembly simplificada;
    - compilar os programas no código de máquina;
    - simular a execução do programa no processador.
- Acréscimo da possibilidade de gerar projectos VHDL com alguns dos componentes do processador omitidos. Os componentes a omitir são escolhidos pelo professor para serem desenvolvidos posteriormente pelos alunos.
- Geração de ficheiros de teste para os alunos poderem verificar componentes omitidos inicialmente e desenvolvidos por eles.

O Processador MIPS32 Single Cycle

Especificação

Para as fases iniciais, a selecção de um processador para o ensino concorrente da lógica de programação e de conceitos de arquitectura de computadores deve facilitar o estabelecimento de relações entre as abstracções lógicas necessárias à programação e a implementação dessas abstracções em hardware.

O processador MIPS Single Cycle de 32 bits possui uma arquitectura já bastante conhecida pelos alunos de MIEET e MIECT da Universidade de Aveiro. Este processador é utilizado como base na disciplina de arquitectura de computadores leccionada nesta academia a ambos cursos, o que faz logo ao princípio um motivo de adopção para desenvolvimento em VHDL, visto que é um processador com que os alunos encontram-se familiarizados.

Este processador possui ainda outras vantagens que levaram a ser escolhido como o primeiro processador a ser desenvolvido. Entre elas o facto de possuir uma arquitectura simples e intuitiva, sendo possível dar a conhecer ao aluno a necessidade das partes que o constituem, a sua interligação e contexto numa perspectiva global da arquitectura do processador.

De seguida, abordaremos a construção em VHDL de forma estruturada, decompondo o processador em cinco fases funcionais: *Instruction Fetch*, *Instruction Decode*, *Execution*, *Data Memory* e *Write Back*. Por fim, a unidade de controlo (*Control Unit*) que opera sobre todas estas fases gerindo as operações a executar em cada uma delas.

A arquitectura

A arquitectura do processador MIPS é baseada nos três tipos de instruções de 32 bits cada: Tipo-R, Tipo-J e Tipo-I, fazendo com que a arquitectura desenvolvida possua um *datapath* de 32 bits consistente com a dimensão das instruções (Tabela 2). No desenvolvimento deste projecto não foi implementado o Tipo-J, no entanto facilmente é acrescentada essa possibilidade, podendo ser uma das possíveis actividades propostas a realizar pelo aluno. Inicialmente é pretendido que a arquitectura implementada suporte um pequeno conjunto de instruções assembly MIPS: Load Word (LW),

Tipo-R						
Campo	Opcode	Rs	Rt	Rd	Shamt	Funct
Bits	31-26	25-21	20-16	15-11	10-6	5-0
Tipo-I						
Campo	Opcode	Rs	Rt	Address		
Bits	31-26	25-21	20-16	15-0		
Tipo-J						
Campo	Opcode	Target				
Bits	31-26	25-0				

Tabela 2 - Formato de instruções MIPS

Store Word (SW), Branch on Equal (BEQ), ADD (soma com overflow), SUB (subtracção com overflow), AND (and bit-a-bit), OR (or bit-a-bit) e Set-Less-Then (SLT). Deste modo pequenos programas assembly podem ser escritos para teste da arquitectura desenvolvida.

O sinal de relógio

A implementação de um processador deste tipo implica que, tal como o nome indica, num ciclo de relógio sejam efectuadas as fases de busca da instrução, a sua descodificação, execução ou acesso à memória e armazenamento dos dados produzidos. No entanto, como se pode verificar ao construir esta arquitectura, não é de todo possível num único ciclo de relógio efectuar todas as sincronizações necessárias. De facto um período completo de um sinal de relógio só possui dois flancos de transição (flanco ascendente e descendente) o que não é suficiente para sincronismo do PC, leitura da instrução da memória, escrita e possível leitura da registos síncronos, assim como a escrita dos resultados produzidos na memória de dados. Deste modo o ciclo de relógio foi dividido por dois, podendo assim usufruir de 4 flancos de sincronização distribuídos pelos vários componentes da arquitectura que os necessitam. Foi atribuído o nome de *CPUClock* ao sinal de relógio que por cada ciclo, efectua uma instrução e *MemClock* ao sinal de relógio que possui o dobro da frequência de *CPUClock*, sendo este associado às leituras e escritas das memórias (Figura 16).

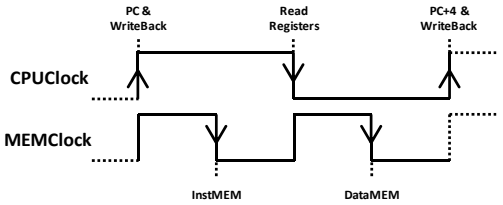


Figura 16 - Diagrama representativo dos acessos efectuados nos flancos de relógio de *CPUClock* e *MEMClock*

Fase Instruction Fetch

A primeira fase da execução no MIPS é a leitura da memória da instrução que se pretende executar (*IF – Intruction Fetch*). Esta operação de leitura é iniciada quando o *Program Counter* (*PC*), registo de 32 bits libera um endereço ao bloco *Instruction Memory* no flanco ascendente do *CPUClock*. O PC é incrementado de modo a indicar o próximo endereço de acesso à memória para a instrução seguinte (*PC+4*). No bloco *Instruction Memory*, lê-se da memória a instrução apontada pelo endereço recebido. Esta operação de leitura de memória é sincronizada com o flanco descendente de *MEMClock* e apenas efectuada quando *CPUClock* possui o valor '1' (ver transição InstMEM na Figura 16).

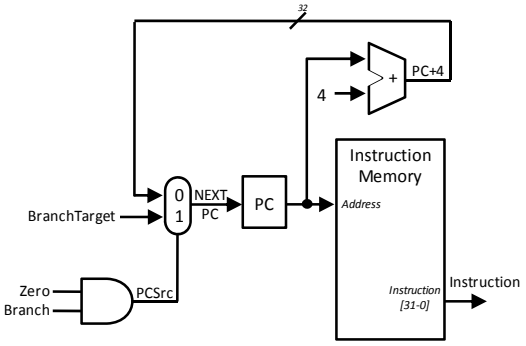


Figura 17 - Diagrama da fase de *Intruction Fetch*

No caso de instruções de salto (Branch), o endereço da instrução seguinte poderá ser substituído pelo endereço gerado pela situação de salto. Para que tal aconteça com sucesso, é necessária lógica adicional que calcule o endereço de salto. Um multiplexer de duas entradas de 32 bits selecciona qual das suas entradas deve ser colocada à saída através do resultado gerado por uma porta lógica AND que avalia se a condição de salto é satisfeita.

A memória de instruções foi implementada com base nos blocos de memória embutida BlockRAM, disponíveis em FPGAs da família Spartan-3 [6]. Esta memória foi inicializada já com instruções que possibilitem posteriormente a análise dos resultados obtidos em simulação e detecção de erros [30,31].

Um diagrama de blocos desta fase é visualizado na Figura 17.

Instruction Decode

Quando a instrução é lida e colocada no barramento de saída do bloco *Instruction Memory*, os vários campos que constituem a instrução são processados. O *Opcode* da instrução é lido pela unidade de controlo do *datapath* e o campo *Funct* lido pela unidade de controlo da *ALU* (Arithmetic Logical Unit). Nesta fase a unidade de controlo (*Control Unit*) após ler o *Opcode* da instrução dispõe à sua saída todos os sinais de controlo do datapath (descrito mais detalhadamente à frente).

Os campos da instrução correspondentes aos registos que se pretendem ler/escrever são lidos pelo módulo *File Register*. Na implementação efectuada, as leituras de registos são assíncronas (podendo ser efectuadas de forma síncrona, ver transição *ReadRegisters* na Figura 16) e a escrita de dados no registo de destino é síncrona com o flanco ascendente de *CPUClock* e apenas executada quando o sinal *RegWrite* se encontra activo.

Os campos lidos, cada um com 5 bits, endereçam registos de 32 bits que são lidos e colocados na saída do módulo. Existem 32 registos (r0-r31), todos disponíveis para guardar valores gerados pelas instruções à excepção do registo r0 que possui sempre o valor zero, norma implementada na arquitectura MIPS, visto que é um valor muitas vezes usado durante a execução de instruções.

O módulo *File Register* possui então quatro entradas: correspondentes aos endereços de dois registos de leitura e um de escrita, um porto de entrada de 32 bits dos dados a escrever e duas saídas de valores contidos nos dois registos lidos: *RegData1* e *RegData2*, ambos de 32 bits. Este módulo foi desenvolvido em VHDL de modo a que após sua síntese, os registos são implementados em LUTs (*LookUp Tables*).

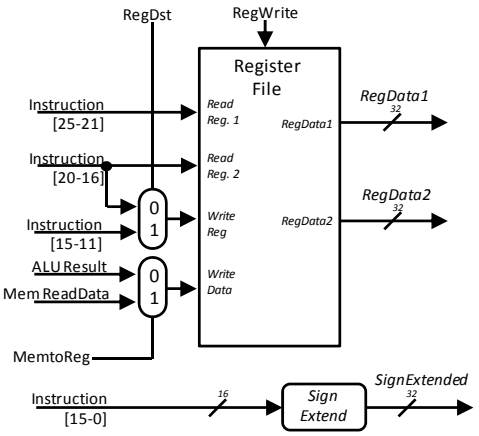


Figura 18 - Diagrama de blocos da fase de *Intruction Decode*

Os 16 bits menos significativos da instrução passam pelo *SignExtended*, módulo que replica o bit mais significativo de forma a gerar uma saída de 32 bits em complemento para 2. Este valor, servirá posteriormente para cálculo de endereços de salto, acesso à memória para leitura/escrita ou leitura de valores imediatos em instruções contendo valores imediatos.

O diagrama de blocos da Figura 18, correspondente a esta fase, representa o descrito nos parágrafos anteriores.

Execution

Após a descodificação da instrução na fase anterior, dá-se início à fase *Execution* que produz todos os cálculos necessários. Os componentes nesta fase são a *ALU*, *ALU Control* e os blocos necessários para cálculo do endereço de salto: um somador de 32 bits e bloco combinatório de deslocamento *SLL2*. Na unidade de *ALU* efectuam-se os deslocamentos, cálculos aritméticos e lógicos e utiliza-se o somador de 32 bits para determinar o endereço relativo de salto duma instrução *branch*. Para efectuar o cálculo do endereço é necessário antes efectuar um deslocamento lógico à esquerda de dois bits do sinal *SignExtended* e depois usar esse resultado e o valor de *PC+4* como entradas do somador de 32 bits. O diagrama de blocos da Figura 19 demonstra a arquitectura desta fase.

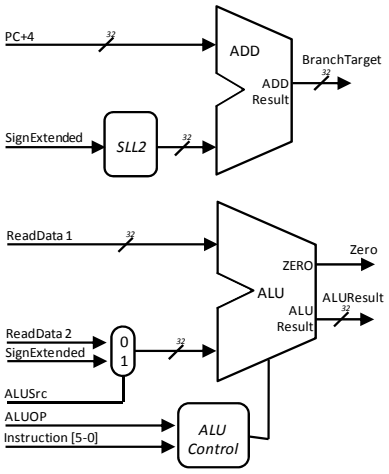


Figura 19 - Diagrama de blocos da fase *Execution*

A ALU possui 3 entradas, sendo duas correspondentes aos valores dos registos no caso de instruções do Tipo-R ou o valor do primeiro registo e os 32 bits do *SignExtended* para instruções do Tipo-I. A selecção da segunda entrada é efectuada através de um multiplexer controlado pela unidade de controlo. A terceira entrada é correspondente aos bits de controlo. Estes bits são gerados pela *ALU Control* que analisa os 6 bits do campo *Funct* das instruções e os 2 bits denominados de *ALUOP* gerados pelo *Control Unit*, decifrando assim que tipo de operação se pretende efectuar na *ALU*. A saída da *ALU Control* é um sinal de 3 bits que permitem gerar uma das combinações mostradas na Tabela 3.

Instrução	Tipo de instrução	ALUOP	Operação a executar	Campo Funct	Operação da ALU	Controlo da ALU
LW	Tipo-I	00	LOAD WORD	XXXXXX	ADD	010
SW	Tipo-I	00	STORE WORD	XXXXXX	ADD	010
BEQ	Tipo-I	01	BRANCH EQUAL	XXXXXX	SUBTRACT	110
ADD	Tipo-R	10	ADD	100000	ADD	010
SUB	Tipo-R	10	SUBTRACT	100010	SUBTRACT	110
AND	Tipo-R	10	AND	100100	AND	000
OR	Tipo-R	10	OR	100101	OR	001
SLT	Tipo-R	10	SET ON LESS THAN	101010	SET ON LESS THAN	111

Tabela 3 - Sinais gerados pelo ALU Control em função do ALUOP e Funct recebidos

O uso de VHDL, fornece grandes vantagens na descrição das condições que se pretendem verificar, usando declarações *CASE-IS* ou *IF-THEN-ELSE* pode-se mesmo quase transcrever a leitura da Tabela 3, exemplo disso é a Figura 20 que representa a implementação usada em VHDL para o módulo *ALUControl*.

```
process(Funct, ALUOp)
begin
      case ALUOp is
      when "00" => Output <= "010";
      when "01" => Output <= "110";
      when others =>
            case Funct is
            when "100100" => Output <= "000";
            when "100101" => Output <= "001";
            when "100000" => Output <= "010";
            when "100010" => Output <= "110";
            when others   => Output <= "111";
      end case;
      end case;
end process;
```

Figura 20 – Código VHDL do módulo *ALUControl* usando declarações do tipo *CASE-IS*

A *ALU* possui dois sinais de saída, em que num coloca o resultado da operação aritmética/lógica efectuada (*ALUResult*) e noutro um sinal de um bit

chamado *Zero*, que fica activo quando o resultado da ALU for igual a zero. Este último sinal é ligado à porta lógica AND que recebe também um sinal do *Control Unit* denominado de *Branch* (activo numa situação de salto conforme ilustrado na Figura 23). Quando estes dois sinais estão activos a condição de salto é validada e o valor do próximo endereço de memória de instruções que deve ser lido é o apontando pela instrução de salto.

Data Memory

A fase *Data Memory*, lê ou escreve valores da ou para a memória. Esta fase numa situação de *Branch* não produz qualquer tipo de efeito. A *Data* Memory (Figura 21) recebe dois valores à entrada correspondentes aos dados a serem escritos (*WriteData*) e o endereço de memória ao qual se pretende aceder (*Address*). Este módulo possui ainda um porto de saída no qual coloca o valor lido da memória (*ReadData*).

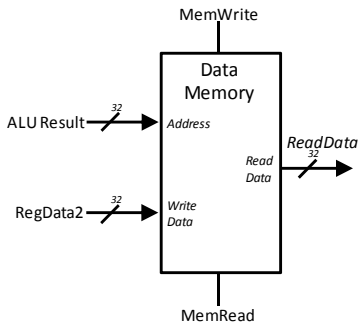


Figura 21 - Diagrama do bloco da fase de *Data Memory*

Dois sinais separados, *MemWrite* e *MemRead*, são usados para garantir a escrita ou a leitura da memória, sendo esta acção sincronizada com o *MEMClock* e *CPUClock* (ver transição *DataMem* na Figura 16).

Para ler da memória o sinal *MemRead* encontra-se activo e *MemWrite* desactivo. Para escrita estes sinais encontram-se invertidos, sendo por isso possível, se pretendido, uma optimização de recursos usando apenas um bit de controlo para escrita e leitura. Durante o desenvolvimento, uma pequena parcela de memória foi inicializada para efeitos de simulação e verificação de erros. A implementação deste tipo de Memória foi feita em módulos BlockRAM, usufruindo das vantagens das estruturas disponibilizadas pela FPGA. Em VHDL uma BlockRAM pode ser instanciada directamente ou usando uma sequência de instruções VHDL que a ferramenta de síntese (XST da Xilinx) consegue interpretar como sendo objectivo do projectista a implementação em BlockRAM do componente descrito. De seguida encontra-se uma parte do código que demonstra a descrição em VHDL da memória de dados (Figura 22).

```
signal Memory : T_DataMem := DataMemory;

process (Clock, Enable)
begin
   if FALLING_EDGE(Clock)  then
    if MemRead = '1' and Enable = '0' then
       ReadData <= Memory(CONV_INTEGER(Address));
    elsif MemWrite = '1' and Enable = '0' then
       Memory(CONV_INTEGER(Address)) <= WriteData;
    end if;
   end if;
end process;
```

Figura 22 – Código VHDL que descreve a implementação de uma memória

Write Back

Nesta fase de uma instrução MIPS, o resultado produzido em fases anteriores como *Execution* ou *DataMemory* pode ser escrito num registo. Um multiplexer determina qual dos resultados obtidos deve ser escrito no registo de destino. Esta operação de selecção é gerida pelo *Control Unit* que através do sinal *MemToReg* efectua o controlo sobre este multiplexer. As instruções envolvidas nesta fase são operações de Registo-Registo ou *Loadword*. Para operações Registo-Registo, o resultado da ALU é passado para escrita sendo necessário para isso o bit de selecção do multiplexer conter o valor ‘0’ e para valores lidos da memória, encontra-se no sinal de selecção o valor activo “1”, escrevendo assim no registo o conteúdo do sinal *ReadData* (pode-se visualizar o diagrama desta situação na Figura 23).

Control Unit

Em todas as fases do MIPS, existem sinais de controlo como já referidos anteriormente. Estes, estão descritos na Tabela 4, e possuem a capacidade de alterar o funcionamento do *datapath* dependente da instrução que se pretende executar.

A unidade de controlo (*Control Unit*) recebe à entrada o *Opcode* da instrução e gera na sua saída os sinais necessários para controlo de registos, memória, multiplexers e *ALUControl*.

Como só depende do *Opcode*, em cada ciclo de relógio os sinais de controlo apenas podem possuir o valor ‘1’, ‘0’ ou ser irrelevantes (*don’t-care*) como se pode verificar na Tabela 4 - . No desenvolvimento em VHDL, o módulo

Control Unit foi construído utilizando *CASE-IS* declarando de uma forma explícita todos os sinais gerados para cada *Opcode* de entrada.

Ao combinar todas as fases anteriormente descritas obtém-se o *datapath* da arquitectura pretendida para um conjunto reduzido de instruções MIPS de 32 bits (ver Figura 23).

Fase do MIPS RISC	Sinais de controlo	Efeito dos sinais
Instruction Fetch	PCSrc	O valor do PC é substituído pelo endereço calculado pela instrução de salto ou pelo valor de PC+4.
Instruction Decode	RegDst	O número do registo de destino provém do valor dos bits 15-11 ou dos bits 20-16 da instrução.
	RegWrite	O registo de destino escolhido é escrito com o conteúdo do sinal ligado à entrada do porto WriteData.
Execution	ALUSrc	O segundo operando da ALU é os 16 bits menos significativos da instrução extendidos de sinal até 32 bits. Caso contrário é o valor do segundo registo lido (ReadData2).
Data Memory	MemWrite	O conteúdo da memória apontado pelo endereço é escrito com o valor à entrada do porto WriteData.
	MemRead	O conteúdo da memória apontado pelo endereço é lido e colocado no porto de saída (ReadData).
Write Back	MemtoReg	O valor que se encontra para escrita no registo provém da memória ou do resultado da ALU.

Tabela 4 - Sinais de controlo em cada fase e seu efeito quando activos

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp 1	ALUOp 0
Tipo-R	1	0	0	1	0	0	0	1	0
LW	0	1	1	1	1	0	0	0	0
SW	X	1	X	0	0	1	0	0	0
BEQ	X	0	X	0	0	0	1	0	1

Tabela 5 - Valores atribuídos aos sinais de controlo dependendo do tipo de instrução

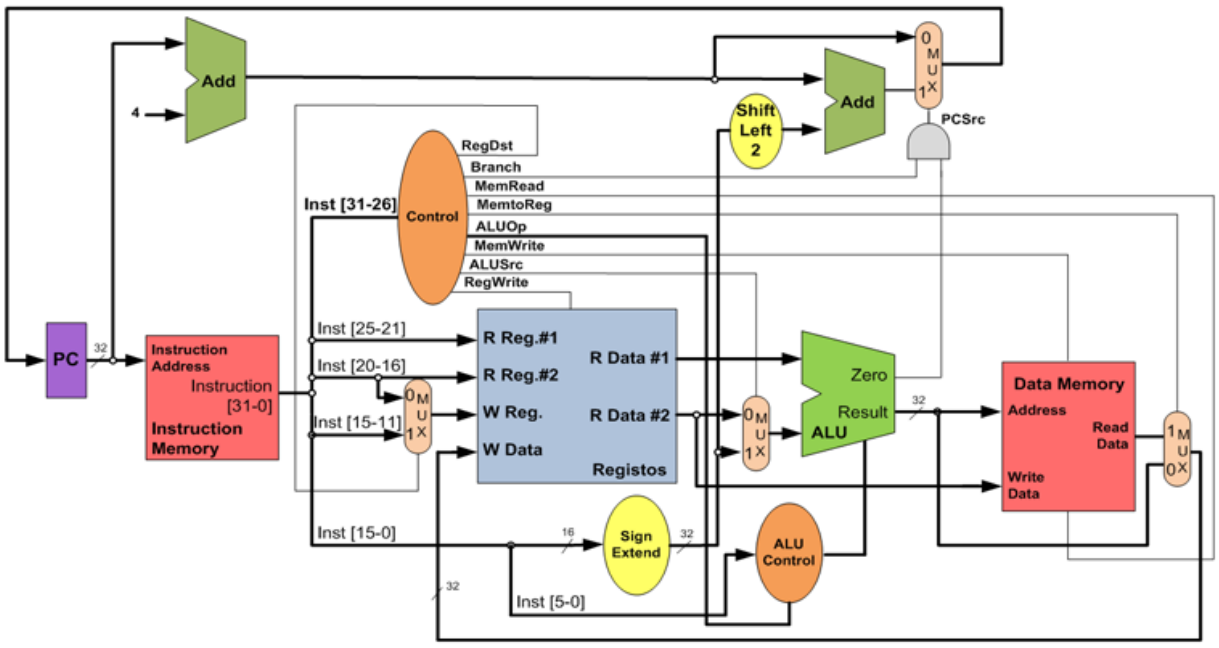


Figura 23 - Arquitectura completa do MIPS Single Cycle de 32 bits

Arquitectura simplificada com a unidade de controlo e a fase de *Write Back* (imagem adaptada de [32])

A MIPS_PACKAGE

Ao longo da construção do processador, foi verificado o constante uso de variáveis comuns em vários blocos. A dimensão do barramento do processador MIPS32 é, tal como o nome indica, de 32 bits. Verificada esta situação é de todo o agrado implementar uma package, que possua algumas constantes da arquitectura e inicialização de valores.

A criação deste ficheiro de declaração de variáveis globais contribuiu para a optimização do processo de síntese por parte da ferramenta Xilinx e ao mesmo tempo, optimizou e facilitou a construção dos vários módulos. A MIPS_PACKAGE possui declarado a dimensão da arquitectura implementada (32 bits), a dimensão do barramento de endereços utilizado (8 bits), as dimensões das memórias implementadas (64 palavras de 32 bits cada) e a inicialização das memórias com instruções e dados.

Outros processadores implementados

O processador projectado foi também alvo de testes e experiências na disciplina de Modelação e Síntese de Processadores (opção de 5º ano de MIEET e MIECT) no ano lectivo 2007/08. Este foi sintetizado e implementado na placa *Spartan 3 Starter Board* da *Digilent Inc* [33] na com ligeiras alterações de modo a ser possível efectuar a leitura de 8 bits (correspondentes aos 8 bits menos significativos) de dados dos switchs para introdução de valores e escrita do resultado final em LEDs, iluminando 8 LEDs correspondentes aos 8 bits menos significativos do resultado pretendido para o registo r8. Isto foi possível usando o bit 7 do resultado da ALU para decisão de escrita para os LEDs. Quando a '1' escreve na memória de dados e também para os LEDs da placa, iluminando-os.

Após o desenvolvimento deste processador, o qual denomino de MIPS32 Single Cycle Basic, outras versões foram desenvolvidas com o intuito de saber até que ponto é possível adaptar este processador como base para novos projectos, suporte de novos conjuntos de instruções ou aquisição de competências para construção de novas arquitecturas projectadas e implementadas pelos alunos.

Muito facilmente adicionaram-se novas funcionalidades como o suporte a um conjunto mais alargado de instruções, tais como, leitura e operações com imediatos (ADDi, ORi, ANDi, SLTi e LUI) e deslocamentos lógicos à esquerda (SLL) e à direita (SRL). Estas funcionalidades podem ser encontradas no processador MIPS32 Single Cycle Advanced.

Ainda no âmbito da disciplina de Modelação e Síntese de Processadores, em união de esforços com o colega Rui Teixeira Sousa, foi desenvolvido um

processador MIPS de 32 bits Multithread com uma arquitectura pipelined de três estágios com capacidade de efectuar uma operação aritmética e de acesso à memória em simultâneo (ver anexo 2).

Análise do Sistema Implementado

O processador MIPS32 Single Cycle com um conjunto de instruções reduzido permite ter uma primeira aproximação à organização e funcionamento de arquitecturas de computadores desde o nível mais baixo – hardware, a um nível mais alto – software, o que depois de estudados e adquiridos os conceitos básicos de desenvolvimento de sistemas digitais permite lançar novos desafios ao aluno.

A possibilidade de desenvolvimento de pequenos módulos que adicionem funcionalidades ao processador é um ponto a favor da arquitectura implementada pois permite ao aluno expandir as capacidades do processador.

O facto de este processador ser um dos exemplos em Patterson e Hennessy [22], constitui uma vantagem para os alunos, visto que podem consultar o livro e efectuar o paralelismo entre o que é explicado e o processador implementado. Da mesma forma a implementação de novas arquitecturas, se for iniciativa ou tarefa do aluno, torna-se mais intuitiva e familiar.

Outra vantagem da escolha deste processador é que, já sendo conhecido pelos alunos, o professor nas aulas teóricas poderá utilizá-lo como exemplo para abordar os mecanismos de desenvolvimento em VHDL de sistemas reconfiguráveis e aplicar esses conhecimentos no laboratório, dando oportunidade aos alunos de por eles próprios (re)construírem alguns módulos omitidos ou arquitecturas completas numa fase mais avançada.

Protocolo de comunicação

Encontrando-se o processador pronto, a análise dos resultados obtidos por simulação ou mesmo da implementação física e visualizando através de LEDs, torna-se demasiada complexa e exigente de tempo, no primeiro caso, devido à quantidade de informação de valores a analisar, no segundo, devido à pouca informação que se pode obter e analisar ao mesmo tempo. Surgiu então necessidade de conseguir criar um mecanismo de comunicação com o processador de modo a facilitar a obtenção e análise dos valores originados pelo mesmo.

Especificação

A placa DETIUA-S3 possui uma bitstream por omissão que implementa um protocolo de comunicação, desenvolvido pelo seu criador Mestre Manuel Almeida em conjunto com o Eng. Bruno Pimentel, que possibilita a comunicação entre um computador e a placa via USB.

O protocolo implementado consiste numa máquina de estados

Figura 12) que permite ao utilizador usufruir da ferramenta PBM para interagir com a memória flash da placa. Assim o utilizador pode ler e escrever na memória novas bitstreams (na zona de memória atribuída ao utilizador, ver Figura 10 do capitulo anterior relativa às zonas da memória flash) ou dados para ou procedentes do projecto implementado.

O facto de haver a necessidade de interagir e comunicar com o processador desenvolvido e de já existir uma base de comunicação com a placa DETIUA-S3, levou à ideia de procurar criar novos estados na máquina de estados do protocolo implementado de modo a conseguir comunicar com o processador podendo efectuar operações de gestão do mesmo, leitura e escrita de dados nas memórias do processador, assim como leitura de sinais internos da arquitectura do processador.

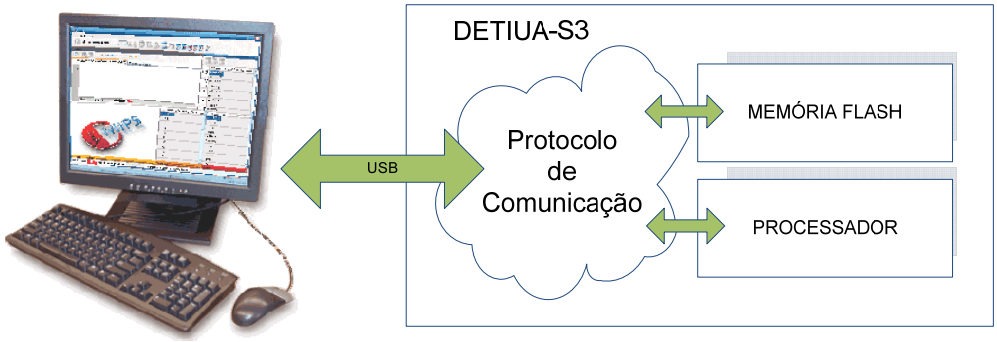


Figura 24 – Diagrama de comunicações do protocolo implementado

Assim foi projectada uma evolução específica do protocolo existente, que permite efectuar as operações básicas sobre um processador (executar ciclos de relógio, efectuar reset ao processador, ler valores dos registos, sinais e memórias e guardar os valores lidos na memória flash) mas mantendo as mesmas capacidades que o protocolo possui (ver Figura 24).

Cinco novos estados foram adicionados ao protocolo de comunicação: MIPS Get Data, MIPS Read FileRegister/DataMemory (modo normal ou seguro), MIPS Erase InstMEM/DataMEM, MIPS SET Instruction/Datamemory e

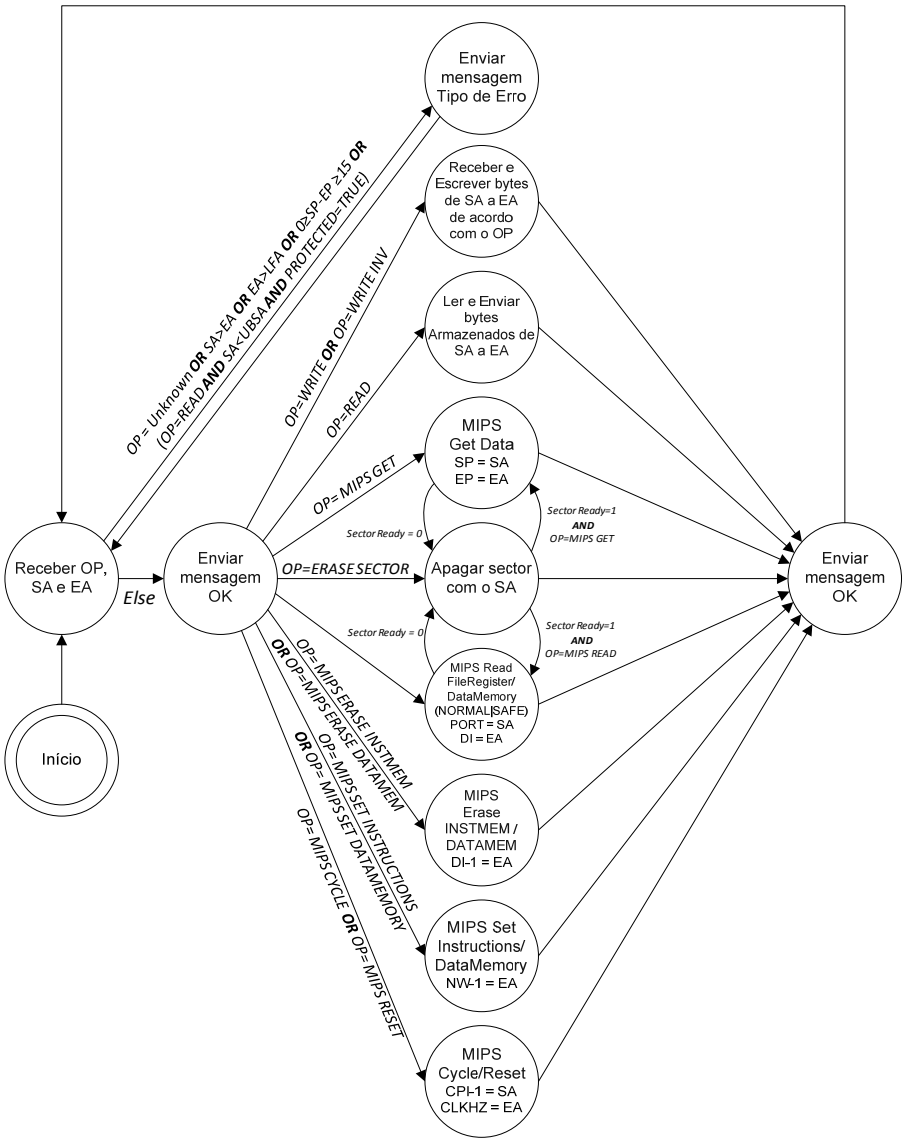
MIPS Cycle/Reset (ver Figura 25) que possibilitam um total de onze novas operações dedicadas à comunicação e gestão do processador MIPS (ver Tabela 6).

O processo de comunicação inicia-se quando é enviada uma mensagem pelo computador para a placa DETIUA-S3. O protocolo de comunicação, implementado juntamente com o processador na FPGA, recebe a mensagem contendo uma operação, descodifica-a e executa-a.

Existem diversas operações permitidas (ver Tabela 6). A título de exemplo, numa operação com o processador de leitura de dados (operação MIPS READ DATAMEMORY) o protocolo gera sinais para o processador podendo ler dados do mesmo e guardar na memória flash (ou ao contrário, ler da memória flash e transmitir para as memórias do processador). Posteriormente, noutra operação com a placa (operação READ FLASH) o conteúdo da memória flash é lido para o computador, podendo nessa altura descodificar e analisar os dados com a aplicação desenvolvida no âmbito desta dissertação.

Aos módulos de VHDL fornecidos pelo Mestre Manuel Almeida que implementam o protocolo de comunicação por omissão foram efectuadas alterações de forma a contemplarem as entradas e saídas de sinais necessários para comunicação com o processador e implementação dos novos estados, assim como ligeiros melhoramentos ao mecanismo já existente. Devido ao envolvimento com protocolo de comunicação por omissão, foi redigido o tutorial de configuração da placa DETIUA-S3 (anexo 3).

Os dados oriundos do processador são sempre escritos no fim da 3ª zona lógica da memória flash (ver Figura 9). Os endereços de escrita foram calculados de modo a garantir memória suficiente para guardar dados oriundos das três fontes contempladas do processador: Sinais, Registos e Memória de Dados.



Legenda:

OP (Operation) - Parâmetro de 1 byte, para especificar a operação;
SA (Starting Address) - Parâmetro de 3 bytes para especificar o endereço da memória flash inicial para a operação solicitada; válido se estiver entre 0 e **EA**;
EA (Ending Address) - Parâmetro de 3 bytes para especificar o endereço da memória flash final para a operação solicitada; válido se estiver entre **SA** e **LFA**;
LFA (Last Flash Address) - Constante específica da placa, que indica o endereço da última posição da memória flash;
UBSA (User Bitstream Starting Address) - Constante específica da placa, que indica o endereço inicial da segunda zona lógica da memória flash;
SP (Start Port) – Parâmetro de 3 bytes para especificar a partir de que porta é para ler e guardar dados na memória flash.
EP (End Port) – Parâmetro de 3 bytes para especificar qual a última porta que se pretende ler e guardar dados na memória flash.
CPI (Cycles per Instruction) – Corresponde ao número de ciclos de relógio que é necessário para executar uma instrução mips.
NW (Number of Words) – Corresponde ao número de words de 32 bits que se pretende processar.
CLKHZ (Clock Hz) – Parâmetro que permite definir o divisor da frequência de relógio que se pretende usar.
DI (Dimension of INSTMEM/DATAMEM) – Parâmetro que permite definir quantas words de 32 bits o módulo INSTMEM/DATAMEM consegue armazenar.
Sector Ready – Representa uma flag interna que informa se o sector já foi apagado. Para se escrever num sector é necessário primeiro apaga-lo.

Figura 25 – Novo diagrama de estados do protocolo de comunicação

O novo protocolo implementado contempla as alterações e a adição dos novos estados para interacção com o processador MIPS

	Operações / Mensagens	Breve descrição
**	ERASE SECTOR	Apaga sectores de memória flash
**	READ FLASH	Lê da memória flash do endereço para o computador
**	WRITE FLASH	Escreve na memória flash valores transmitidos a partir do computador
**	WRITE INVERSE FLASH	Idêntico a **WRITE FLASH**, mas escreve o byte em ordem inversa na memória flash
*	MIPS CYCLE	Gera ciclos de relógio para o processador
*	MIPS RESET	Idêntico a MIPS CYCLE, só que em estado de RESET
*	MIPS GET DATA	Guarda na memória flash os sinais do processador
*	MIPS READ FILE REGISTER	Guarda na memória flash o valor de todos os registos do banco de registos do processador
*	MIPS READ FILE REGISTER (SAFE)	Idêntico a MIPS READ FILE REGISTER, mas gera um CPI no final da leitura para repor os valores à saída do FILEREGISTER
*	MIPS READ DATAMEMORY	Guarda na memória flash o conteúdo da memória de dados do processador
*	MIPS READ DATAMEMORY (SAFE)	Idêntico a MIPS READ DATAMEMORY, mas gera um CPI no final da leitura para repor os valores à saída da DATAMEMORY
*	MIPS SET INSTRUCTIONS	Escreve na memória de instruções do processador os valores guardados anteriormente na memória flash
*	MIPS SET DATAMEMORY	Escreve na memória de dados do processador os valores guardados anteriormente na memória flash
*	ERASE INSTMEM	Apaga a memória de instruções do processador
*	ERASE DATAMEM	Apaga a memória de dados do processador
**	OP ERROR	Mensagem de operação desconhecida
	OK	Mensagem de operação concluída com sucesso
	JUMPER ERROR	Mensagem de erro indicando que não se encontra colocado o jumper
	ADDRESS ERROR	Mensagem de erro indicando que o conjunto de endereços introduzido não é válido
*	NUMBER OF WORDS ERROR	Mensagem de erro indicando que o número de words introduzidas excede o número de words definido para a arquitectura na MIPS_PACKAGE
*	PORT NUMBER ERROR	Mensagem de erro indicando que a porta é inválida ou gama de portas de dados a ler não é válida
*	UNKNOWN ERROR	Mensagem que indica um erro desconhecido na descodificação da operação

Legenda:
* Novas operações;
** Operações já existentes desde da primeira versão do protocolo, apenas optimizadas em algumas situações que causavam alguma instabilidade no protocolo de comunicação.

Tabela 6 – Operações e mensagens admitidas no protocolo de comunicação implementado.

Integração com o processador

O esquemático final do protocolo (Figura 26) dá a possibilidade de o aluno poder visualizar as várias partes que constituem o protocolo e como o seu processador encaixa no processo de comunicação. As alterações tiveram como objectivo final a criação de um interface simples e perceptível para que o aluno possa ligar o seu processador ao protocolo implementado sem grandes dificuldades.

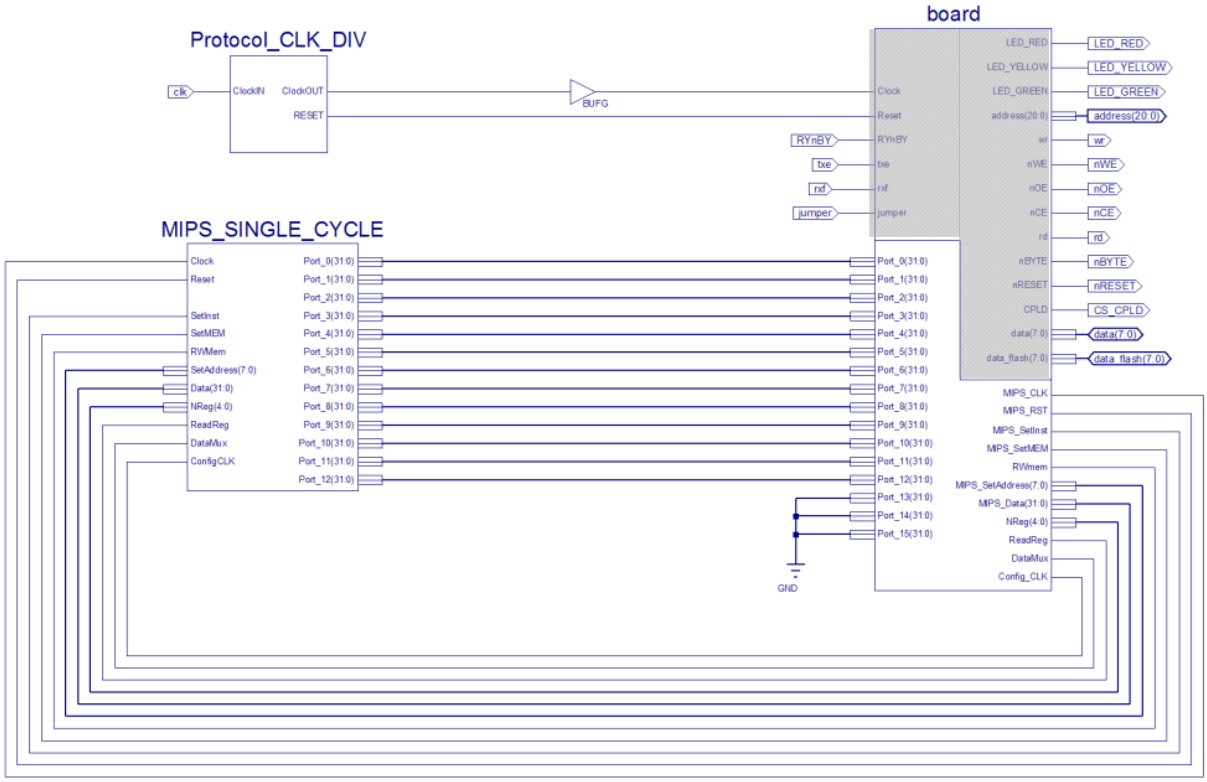


Figura 26 – Esquemático do circuito final
Na figura os blocos relativos ao Divisor de relógio, ao protocolo de comunicação e ao processador desenvolvido.

A placa DETIUA-S3 é equipada com um oscilador de sinal capaz de produzir um sinal de relógio de 80 MHz. No entanto o projecto tem como característica funcionar a 5MHz e isto deve-se ao facto da necessidade de obter frequências mais baixas de modo a conseguir-se operar com a memória flash da placa [16]. O divisor de relógio da Figura 27 possui embutido um DCM (Digital Clock Manager) que tem como entrada o sinal de relógio da placa DETIUA-S3, e à saída o sinal de relógio dividido e o sinal de reset.

O módulo denominado de “board” na Figura 26, é o componente vital para toda a comunicação com a placa DETIUA-S3 com o processador, memória flash e USB e encontra-se dividido em duas partes:

- A superior (ver Figura 26 a escuro) é responsável pela comunicação USB da placa com o computador e a leitura e escrita da memória flash (entre outros não relevantes para o projecto).
- A inferior (ver Figura 26 a branco) é a responsável por efectuar a comunicação com o processador implementado.

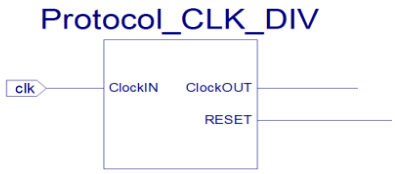


Figura 27 – Divisor do sinal de relógio

Módulo correspondente ao divisor do sinal de relógio usado no protocolo de comunicação

Na parte superior o utilizador não necessita de alterar nada, mas a parte inferior é a parte relevante ao utilizador. Nela pode encontrar os portos de entrada e saída dos sinais necessários. Do lado esquerdo do módulo utilizador pode ligar todos os sinais que pretende ler do seu processador, tendo disponível para esse efeito 16 portas de 32 bits cada. Do lado oposto, lado direito, encontram-se todos os sinais de necessários para controlo do processador (consultar a tabela 5 do manual de utilização do iCmips em anexos para conhecer o significado e efeitos de cada uma destas portas).

Foram ainda desenvolvidos 3 módulos (SetDataMEM, SetInstMEM e SetRegister) com o intuito de serem embutidos num qualquer processador de modo a permitirem a acção externa sobre 3 componentes comuns num processador de arquitectura MIPS (Memória de Dados, Memória de Instruções e Banco de Registos, respectivamente). O código fonte desses módulos (Figura 28) pode ser encontrado juntamente com o projecto base disponibilizado para futuro uso e integração da arquitectura do processador do utilizador caso pretendido (exemplo integração na Figura 29).

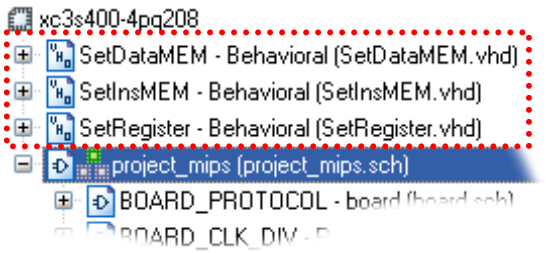


Figura 28 – Modulos de integração no processador

Dentro do rectângulo a tracejado as 3 sources que implementam a interface de controlo da memória de dados, instruções ou banco de registos (respectivamente de cima para baixo)

Estes módulos são constituídos por um mecanismo de multiplexers (ver Figura 29) que permutam os sinais das portas de entradas e de saída, às quais se devem ligar os sinais respectivos de leitura ou escrita de dados. De uma forma geral, os módulos desenvolvidos quando activos isolam os componentes

aos quais estão associados, permitindo ao utilizador obter informações dos mesmos sem que isso corrompa o estado e valores actuais da execução do processador.

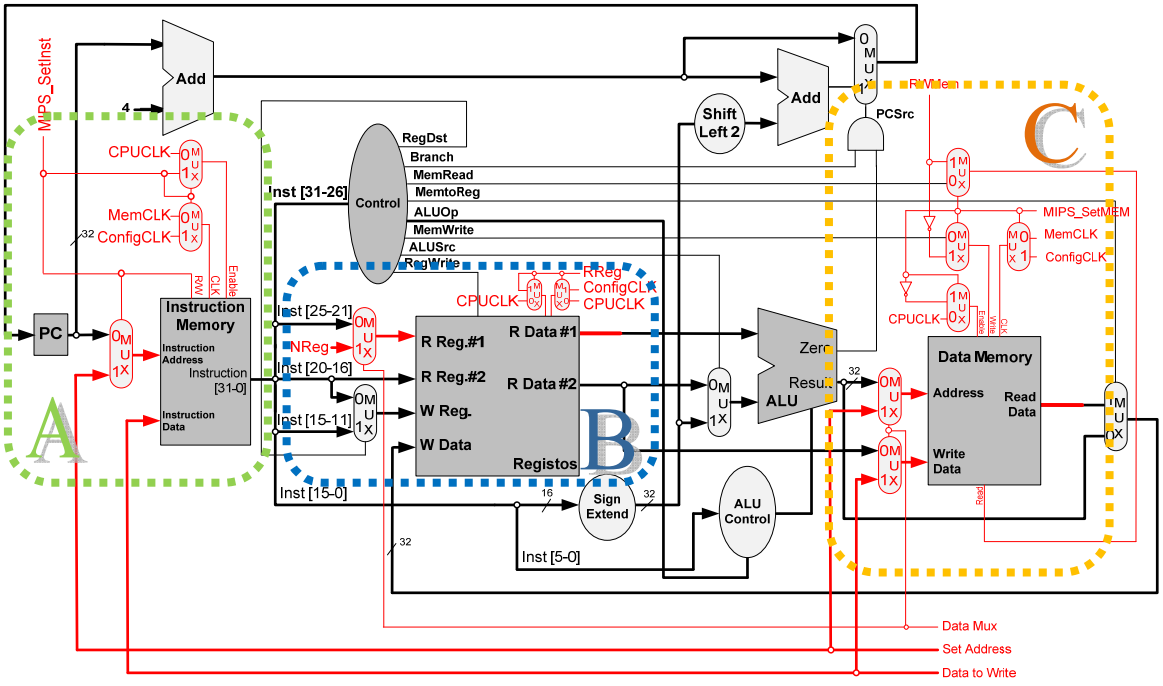


Figura 29 - MIPS Single Cycle Datapath com os módulos de integração

Datapath da arquitectura com os modulos SetInstMEM (área A), SetRegister (área B) e SetDataMEM (área C) adicionados

Na Figura 29 pode visualizar um exemplo de como podem-se implementar os 3 módulos de controlo (áreas A, B e C) que possibilitam o carregamento de instruções (área A) e dados (área C) e leitura do conteúdo dos registos (área B) e memória (área C).

Os componentes são facilmente adicionáveis em qualquer projecto MIPS (ou outro), sendo introduzidos novos portos de entrada à arquitectura do processador para os sinais: **ConfigCLK**, **MIPS_SetInst**, **ReadReg**, **NReg**, **MIPS_SetMEM**, **RWmem**, **DataMux**, **MIPS_Data** e **MIPS_SetAddress**.

Mais informações sobre as operações e seus parâmetros, assim como o mecanismo de funcionamento do protocolo implementado podem ser encontrados no anexo 4.

Análise do Sistema Implementado

O protocolo implementado para além de solucionar algumas das inconsistências verificadas na primeira versão do mesmo, veio a possibilitar a comunicação com o processador implementado.

A disposição dos componentes que constituem o protocolo e seus mecanismos de funcionamento permite ao aluno facilmente compreender a arquitectura de comunicação implementada e fazer uso dela para comunicação com o seu processador.

A utilização do protocolo implementado não implica modificações à arquitectura do processador gerado, o que significa um menor desgaste na procura de soluções para a arquitectura de um processador que contemple o protocolo existente. O aluno possui sempre a decisão de escolher se pretende adicionar os módulos de acesso aos registos ou memórias ou se apenas pretende fazer uso da leitura dos sinais que interligam os componentes da sua arquitectura.

Interface Gráfica (iCmips)

Encontrando-se completa a implementação do processador e do protocolo de comunicação com o mesmo é agora altura de desenvolver uma ferramenta capaz de usufruir das capacidades criadas no protocolo e que possibilite a aquisição, manipulação e análise dos valores gerados pelo processador.

Especificação

O desenvolvimento da GUI (Graphical User Interface) foi feito em Microsoft C# e baseou-se em algumas das funcionalidades de comunicação já implementadas no projecto PBM desenvolvido pelo Eng. Bruno Pimentel. Outra fonte de inspiração para o projecto desta interface foi o software PCSPIM [34], um simulador de processadores que permite a verificação dos valores gerados ao correr um programa assembly num processador MIPS.

Figura 30 – Logótipo da ferramenta de interface iCmips 1.0

O software iCmips 1.0 (Figura 30) foi desenvolvido no intuito de satisfazer a necessidade de possuir uma ferramenta capaz de facilitar a execução de testes ao processador implementado, assim como facilitar a compreensão de valores obtidos à saída dos módulos constituintes da arquitectura implementada.

Assim surgiu o “Eu vejo o MIPS” (do trocadilho da leitura oral em inglês da expressão: “I C *(see)* MIPS”), uma ferramenta de software complementar ao protocolo de comunicação desenvolvido que dá a possibilidade de o utilizador comunicar com um processador de arquitectura MIPS em tempo real, verificando a sua integridade e autenticidade dos resultados produzidos.

Características

O iCmips 1.0 é uma ferramenta de software que possui as seguintes características:

- Suporta todas as operações existentes no protocolo de comunicação entre as quais:
  - Gerar ciclos de relógio para o processador
  - Efectuar Reset ao processador
  - Ler os dados do processador
  - Escrever dados nas memórias do processador
- É adaptativa a interagir com novos processadores desenvolvidos a nível da:
  - Configuração da arquitectura do processador
  - Configuração dos sinais e suas dimensões ligados às portas
  - Configuração de novas instruções MIPS assembly
- Interpretar programas assembly e gerar o seu código máquina;

Para além destas características base, que eram à partida pedidas, outras funcionalidades foram implementadas tais como: auto-detectação da placa DETIUA-S3, um editor de texto para criação de programas assembly com capacidade de highlight de texto assembly, capacidade de carregamento da imagem contendo o diagrama da arquitectura implementada, subjanelas dimensionáveis e dedicadas à visualização de dados oriundos de zonas específicas do processador e ainda a capacidade de guardar as informações de configuração da arquitectura desenvolvida em ficheiros para posterior utilização e rápida configuração da ferramenta.

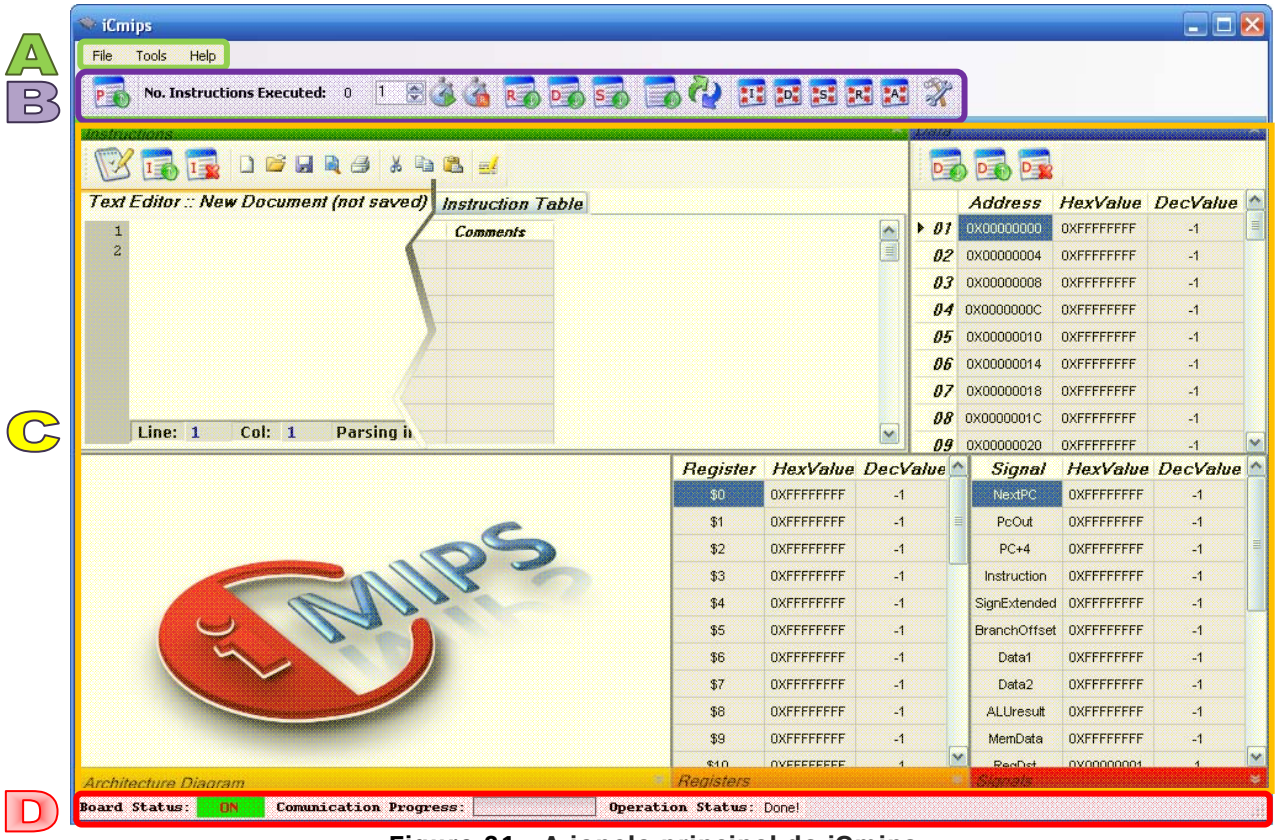


Figura 31– A janela principal do iCmips

A janela principal do iCmips pode ser dividida em quatro áreas distintas (ver Figura 6 – A, B, C e D):

Área A – Barra de Menus;
Área B – Barra de Ferramentas;
Área C – Sub-Janelas de visualização de dados;
Área D – Barra de Estado e Informações.

Na Figura 31 podemos observar quatro áreas que constituem o aspecto do *layout* da aplicação desenvolvida. Na área A o utilizador encontra o acesso a menus contendo as opções diversas da ferramenta (entre as quais comunicação com a placa, configuração do programa e acesso ao manual de utilização). Na área B, respectiva à barra de ferramentas, o utilizar pode aceder aos botões de operações com a placa (selecção de numero de instruções a executar, gerar ciclos de relógio, reset, escrita de conteúdos da memórias do processador, manipulação das subjanelas de dados). A área C é constituída por subjanelas dimensionáveis (código fonte das subjanelas adaptado de [35]) contendo cada uma delas conteúdos específicos a um dos componentes da arquitectura do processador (subjanela da memoria de dados, editor (código fonte adaptado de [36]) e tabela de instruções, registos, sinais e diagrama da arquitectura), sendo estes conteúdos apresentados sob a forma de tabelas. Em baixo é apresentada uma barra de estados, área D, onde é visualizável informação sobre o estado de ligação com a placa, progressão da execução das operações e em que fase encontra-se o protocolo de comunicação.

Para que o programa iCmips funcione correctamente, é necessário antes tomar uma pequena atenção às configurações do software.

Como é compreensível, podem ser implementadas centenas de variantes de processadores de arquitectura MIPS, sendo apenas a imaginação o limite. Assim para que o software iCmips funcione correctamente, não esquecendo que é uma ferramenta para visualização de valores de sinais reais que se encontram na placa DETIUA-S3, é necessário informar a ferramenta de como é que os sinais foram ligados e que tipo de componentes foram usados no processador MIPS implementado.

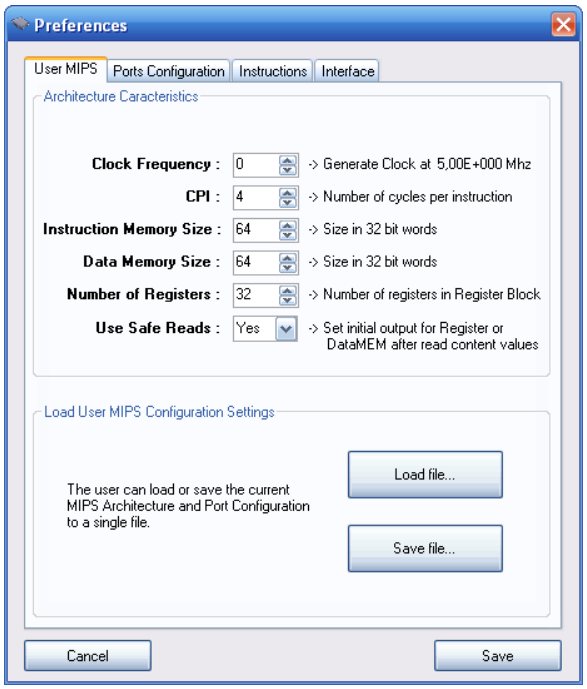


Figura 32 – Painel de configuração da arquitectura MIPS da Janela de Preferências

A janela de Preferências do iCmips (Figura 32) possui 4 painéis de configuração, sendo 3 deles (User MIPS, Ports Configuration e Instructions) dedicados unicamente à discrição das características da arquitectura do processador implementado e um outro (Interface) para fins de configuração de algumas das opções do software iCmips.

O painel de configuração USER MIPS permite ao aluno configurar a ferramenta para funcionar com as características do processador que implementou: número de ciclos necessários para execução de uma instrução (CPI), dimensão do banco de registos e memórias projectadas, entre outras funcionalidades como guardar ou carregar as configurações do programa iCmips num ficheiro para futura utilização e partilha entre colegas e professores para que os mesmos possam replicar os resultados obtidos na ferramenta.

O painel PORT CONFIGURATION possibilita a nomeação e especificação de como o aluno ligou às 16 portas de 32 bits do protocolo de comunicação os sinais existentes na arquitectura do processador desenvolvido. A ferramenta consegue interpretar até 32 sinais de 1 bit por porta.

A ferramenta iCmips 1.0 foi projectada para que seja dinâmica a nível de suporte do número de instruções MIPS. O aluno, no painel INSTRUCTIONS, pode adicionar, editar e remover suporte de instruções MIPS fazendo com que o software seja então capaz de interpretar novas instruções no seu editor de

texto ou instruções já por omissão declaradas mas com uma configuração de campos redefinido pelo utilizador.

Por último, o painel INTERFACE dá a opção do utilizador configurar alguns dos comportamentos e efeitos visuais embutidos na aplicação (como por exemplo auto-arranjo espacial das subjanelas ou cor do texto e fundo do editor de texto).

Mais informações sobre todas as características e capacidades desta ferramenta podem ser consultadas no manual de utilização do software iCmips 1.0 no anexo 4.

Análise do Sistema Implementado

O software desenvolvido proporciona uma das grandes vantagens do sistema projectado. A ferramenta disponibiliza um conjunto de acessos de alto nível para a interacção e depuração de qualquer processador desenvolvido pelo aluno. Numa simples acção de um clique de um botão o aluno tem acesso quase instantâneo à informação dos resultados gerados pelos componentes do processador fisicamente implementado na FPGA.

O facto de a ferramenta possuir um editor de texto que permite a escrita de programas assembly e logo de seguida efectuar a análise lexical e sintáctica da linguagem assembly para depuração de erros e em caso de sucesso gerar o código máquina para o processador desenvolvido, proporciona uma mais-valia na diminuição de tempo despendido para a escrita de programas de teste para o processador, optimizando assim o tempo de aprendizagem da arquitectura de computadores.

Capítulo 4 - Resultados e sua análise

Testes efectuados para cada fase

Processador

Para síntese e simulação deste projecto usou-se a ferramenta ISE Xilinx 9.2.04i [37]. Esta ferramenta possui um simulador integrado (ISE Simulator) no qual permite simular a execução de código assembly (apresentado na Tabela 7) na arquitectura projectada e analisar os resultados obtidos com os valores esperados. Uma parte da memória de dados foi inicializada com alguns valores conforme se pode ver na Tabela 8 (estes valores, dados e instruções, podem ser encontrados na MIPS_PACKAGE).

Nº da Instrução	Endereço de Memória (Hex)	Instrução em código máquina (Hex)	Instrução Assembly (valores em hexadecimal)
1.	00	0X8C010000	lw r1, 0(r0)
2.	04	0X8C030004	lw r3, 4(r0)
3.	08	0X8C040008	lw r4, 8(r0)
4.	0C	0X8C050020	lw r5,20(r0)
5.	10	0X8C070010	lw r7,10(r0)
6.	14	0X8C620000	Loop: lw r2, 0(r3)
7.	18	0X00822020	add r4,r4,r2
8.	1C	0X00611820	add r3,r3,r1
9.	20	0X0065302A	slt r6,r3,r5
10.	24	0X10C7FFFB	beq r6,r7,Loop
11.	28	0XAC040024	sw r4,24(r0)
12.	2C	0X8C080024	lw r8,24(r0)

Tabela 7 - Conteúdo da *Instruction Memory*

Endereço de Memória (Hex)	Conteúdo da Memória de Dados (Hex)
00	0x04
04	0x10
08	0x00
0C	0x08
10	0x01
14	0x02
18	0x03
1C	0x04
20	0x20
24	0x00

Tabela 8 - Conteúdo da *DataMemory*

O código assembly consiste num conjunto inicial de instruções de leitura de valores da memória, seguido de um ciclo de 4 iterações que efectua o somatório dos elementos de um array de valores inteiros. Por fim, o valor do somatório é escrito em memória, e novamente lido. Em seguida são mostrados alguns dos resultados obtidos correspondentes à simulação sequencial das instruções número 6, 7, 9, 11 e 10 (a troca das instruções 10 e 11 foi propositada para efeitos de demonstração das simulações). Deste modo, na Figura 33 pode-se verificar o PC a iniciar com o valor 0x00 na instrução 6 até ao valor 0x10 da instrução 10, voltando novamente a 0x00 devido à instrução 10 corresponder a um *branch taken*, neste subconjunto de instruções simuladas. O subconjunto de instruções escolhido permite visualizar os valores dos sinais no *datapath* para diferentes formatos de instrução.

De seguida apresento os resultados da simulação dos sinais de cada uma das fases e por fim a simulação completa do conjunto de instruções da Tabela 7 observando a evolução do conteúdo do banco de registos.

SIMULAÇÃO DA FASE INSTRUCTION FETCH

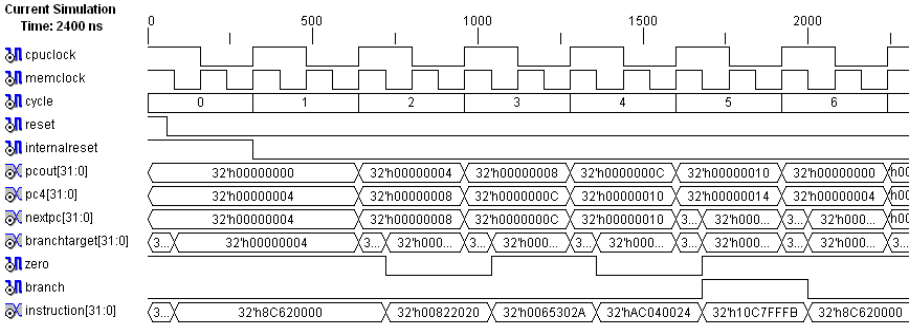


Figura 33 – Diagrama de sinais gerados na fase de Instruction Fetch

SIMULAÇÃO DA FASE INSTRUCTION DECODE

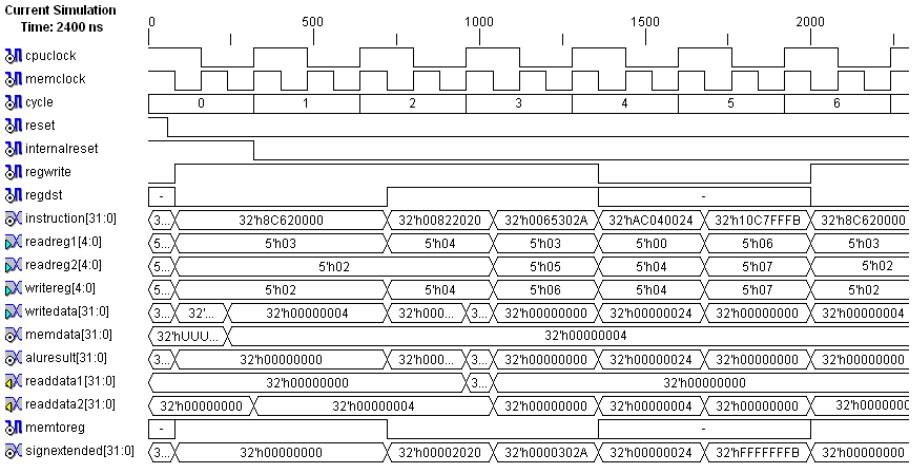


Figura 34 – Diagrama de sinais gerados na fase de Instruction Decode

SIMULAÇÃO DA FASE EXECUTION

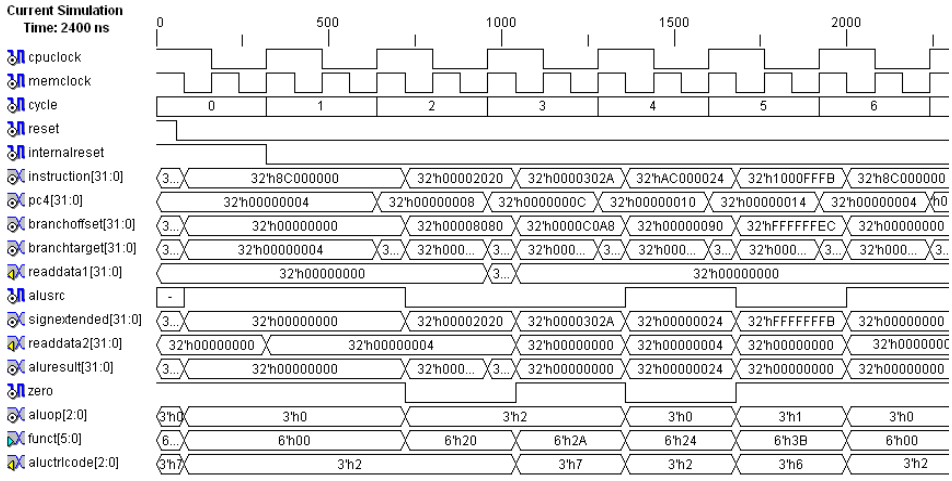


Figura 35 – Diagrama de sinais gerados na fase de Execution

SIMULAÇÃO DA FASE DATAMEMORY E WRITEBACK

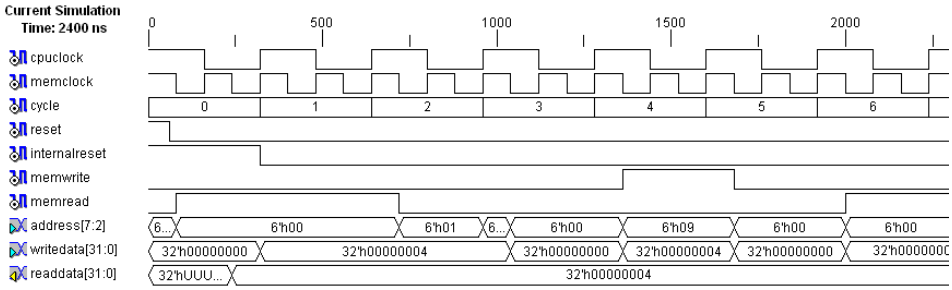


Figura 36 – Diagrama de sinais gerados na fase de DataMemory e WriteBack

SIMULAÇÃO DA UNIDADE DE CONTROLO

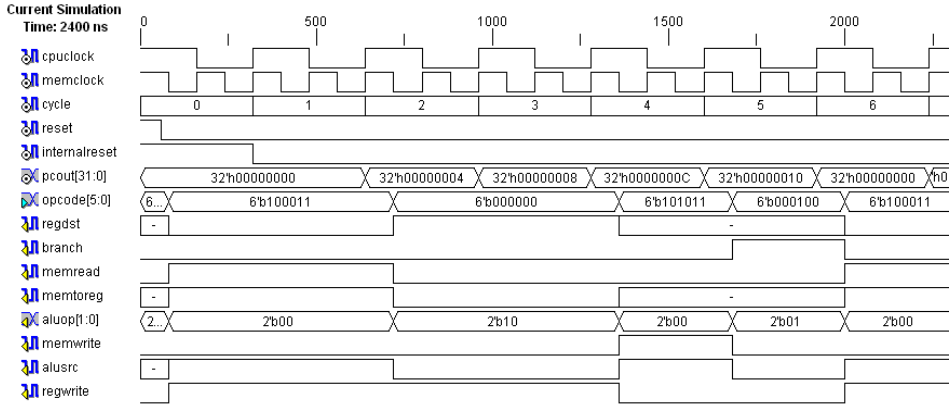


Figura 37 – Diagrama de sinais gerados pela unidade de controlo

SIMULAÇÃO DA ARQUITECTURA DESENVOLVIDA PARA TODAS AS INSTRUÇÕES DA TABELA 7

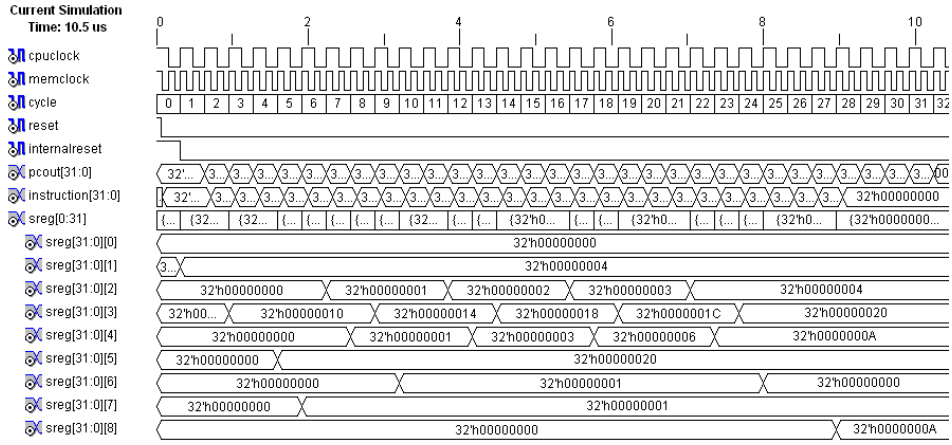


Figura 38 – Diagrama de sinais gerados pela arquitectura implementada

No fim da execução deste trecho de código foi possível verificar que os valores finais correspondem aos pretendidos, demonstrando assim todos os componentes foram bem implementados e as várias fases interligadas produzem o efeito esperado, visto que todos os valores foram devidamente processados.

O processador foi implementado utilizando a ferramenta Xilinx ISE da qual obteve-se as Tabela 9 e 10 contendo a informação sobre o projecto e utilização dos recursos da FPGA Xilinx Spartan 3. A Tabela 10 reflecte apenas os valores estimados pela ferramenta de síntese pois, a placa DETIUA-S3 não possui conectores de interface embutidos (LED's, displays de 7-Segmentos, porta PS/2 ou VGA) e o projecto não possui configuração para mapeamento de ligações aos portos de saída da FPGA fazendo com que a ferramenta remova toda a lógica do circuito implementado durante a optimização do projecto.

Esta primeira fase do projecto, o processador a ser implementado na placa DETIUA-S3 foi unicamente testado por simulação.

MIPS_SINGLE_CYCLE Project Status			
Project File:	MIPS_Single_Cycle.ise	**Current State:**	Programming File Generated
Module Name:	MIPS_SINGLE_CYCLE	• **Errors:**	No Errors
Target Device:	xc3s400-4pq208	• **Warnings:**	34 Warnings
Product Version:	ISE 9.2.04i	• **Updated:**	sáb 3. Maio 15:16:35 2008

Tabela 9 – Tabela sumária do estado do projecto MIPS32 SINGLE CYCLE

Os 34 avisos existentes no projecto são referentes a sinais não ligados, sendo todos os avisos esperados à partida, ou avisos relativos a informações da ferramenta de síntese não relevantes ao projecto desenvolvido.

Device Utilization Summary (synthesis estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	248	3584	6%
Number of Slice Flip Flops	8	7168	0%
Number of 4 input LUTs	560	7168	7%
Number of bonded IOBs	3	141	2%
Number of BRAMs	2	16	12%
Number of GCLKs	2	8	25%

Tabela 10 – Tabela sumária da utilização dos recursos da FPGA Spartan3 na fase de síntese do projecto MIPS32 SINGLE CYCLE

Na Tabela 11 são apresentados os resultados da implementação do mesmo processador MIPS Single Cycle de 32 bits, mas já suportando um maior conjunto de instruções (ver no anexo 4 “MIPS32_Single_Cycle_Basic” e “MIPS32_Single_Cycle_Advanced” na Tabela 8), na placa Spartan3 Starter Board da Digilent Inc [33] que possui uma FPGA Spartan3 (XC2S200) da mesma família da FPGA utilizada na placa DETIUA-S3 (XC2S400). Nesta placa efectuou-se o controlo e depuração do processador através do uso dos switchs, LED’s e displays de 7-Segmentos.

Device Utilization Summary (Implementation Values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	7	3,840	1%
Number of 4 input LUTs	716	3,840	18%
Logic Distribution			
Number of occupied Slices	498	1,920	25%
Number of Slices containing only related logic	498	498	100%
Number of Slices containing unrelated logic	0	498	0%
Total Number of 4 input LUTs	983	3,840	25%
Number used as logic	716		
Number used as a route-thru	11		
Number used for Dual Port RAMs	256		
Number of bonded IOBs	19	173	10%
IOB Flip Flops	8		
Number of Block RAMs	2	12	16%
Number of GCLKs	2	8	25%
Number of DCMs	1	4	25%

Tabela 11 – Tabela sumária da utilização dos recursos da FPGA Spartan3 na fase de implementação do projecto MIPS32 SINGLE CYCLE ADVANCED na placa Spartan 3 Starter Board da Digilent Inc. [33]

Protocolo

O teste dos estados implementados foi efectuado fazendo uso da consola de comunicação do software PBM na qual foram introduzidos os códigos e parâmetros das operações a testar e posteriormente verificados os resultados obtidos com os esperados. Um tutorial de exemplificativo do efectuado pode ser encontrado no anexo 4.

No intuído demonstrar que sinais são produzidos pela interface implementada para o processador, de seguida são apresentados alguns diagramas temporais dos sinais gerados. Os testes são referentes à arquitectura visualizada na Figura 26. Devido ao facto de serem produzidos imensos ciclos de relógios para comunicação USB e para leitura e escrita na memória flash, os sinais apresentados não demonstram a interactividade com estes periféricos.

MIPS CYCLE

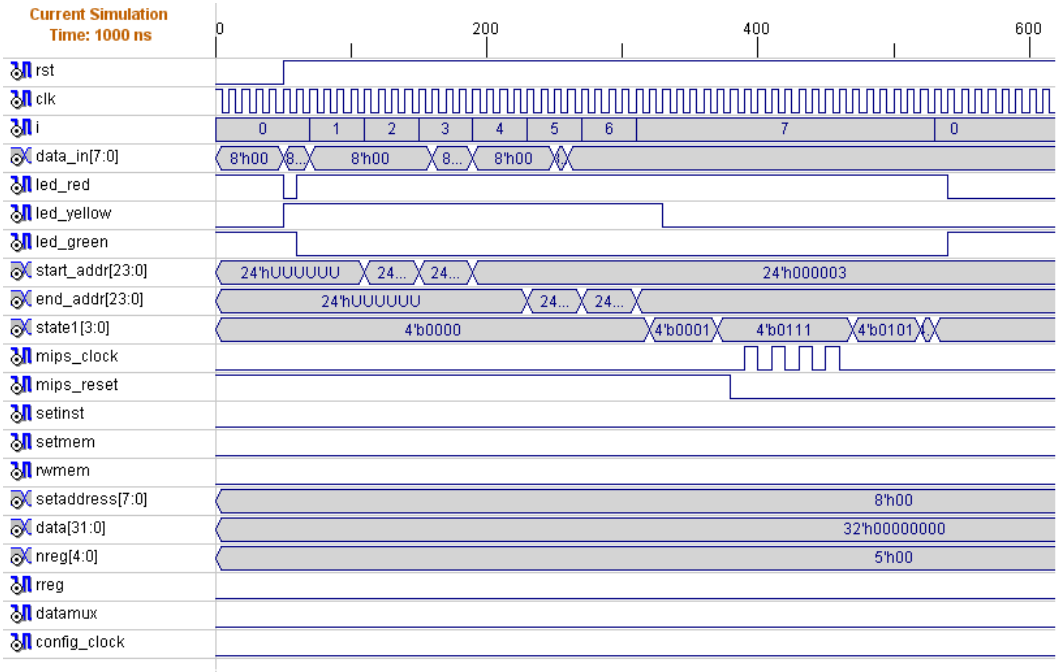


Figura 39 – Diagrama temporal dos sinais gerados numa operação de MIPS CYCLE

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS CYCLE com os parâmetros CPI=4 e CLKHZ = 0;

MIPS RESET

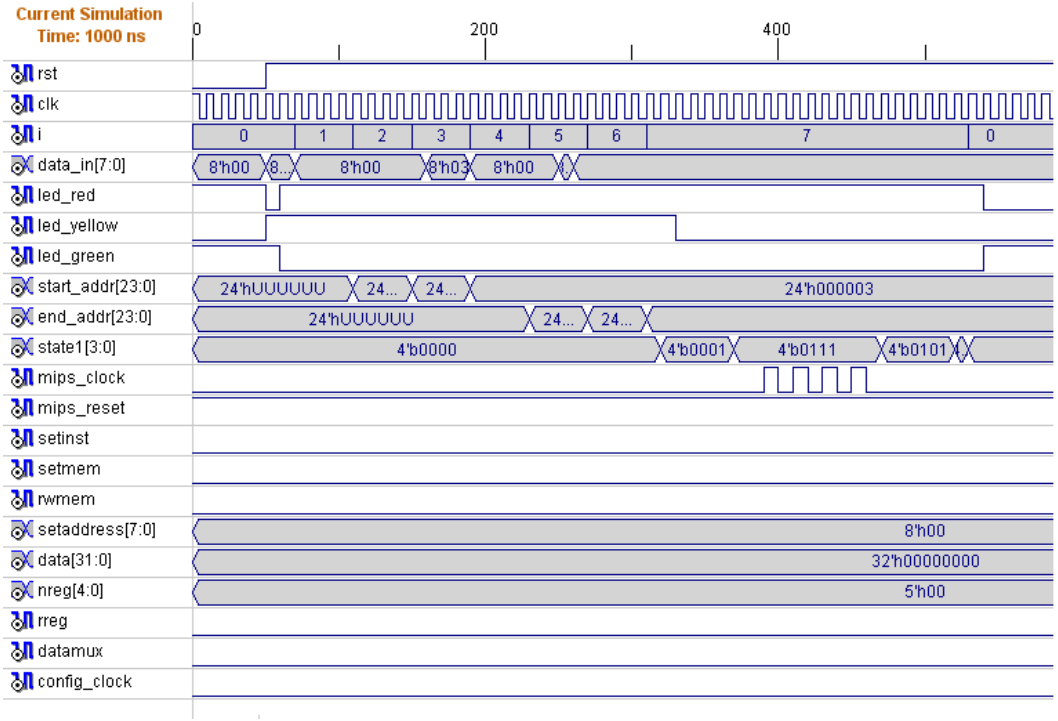


Figura 40 – Diagrama temporal dos sinais gerados numa operação de MIPS RESET

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS RESET com os parâmetros CPI=4 e CLKHZ = 0;

MIPS READ FILE REGISTER

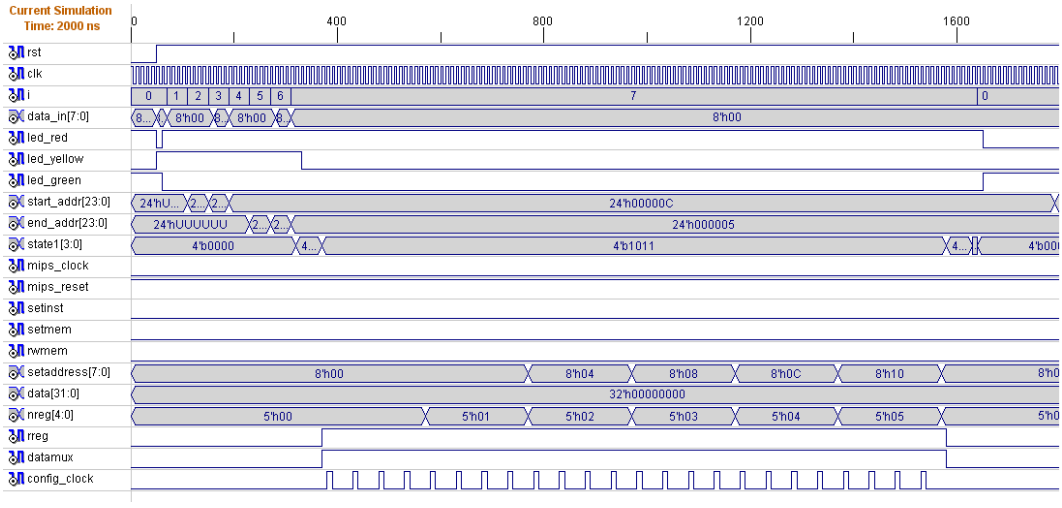


Figura 41 – Diagrama temporal dos sinais gerados numa operação de MIPS READ FILE REGISTER

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ FILE REGISTER com os parâmetros PORT=11 e Nº Registos = 5;

MIPS READ FILE REGISTER (Safe mode)

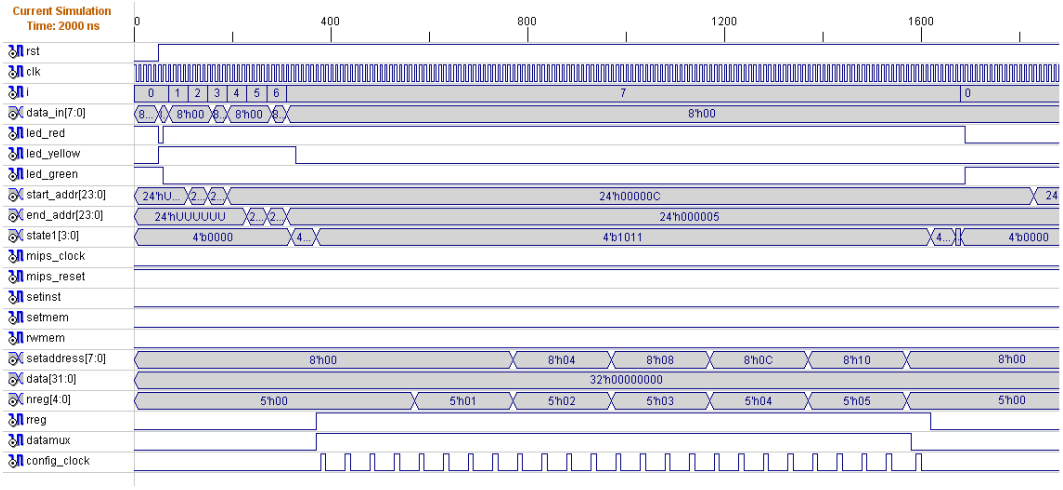


Figura 42 – Diagrama temporal dos sinais gerados numa operação de MIPS READ FILE REGISTER (Safe Mode)

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ FILE REGISTER (Safe) com os parâmetros PORT=11 e Nº Registos = 5;

MIPS READ DATA MEMORY

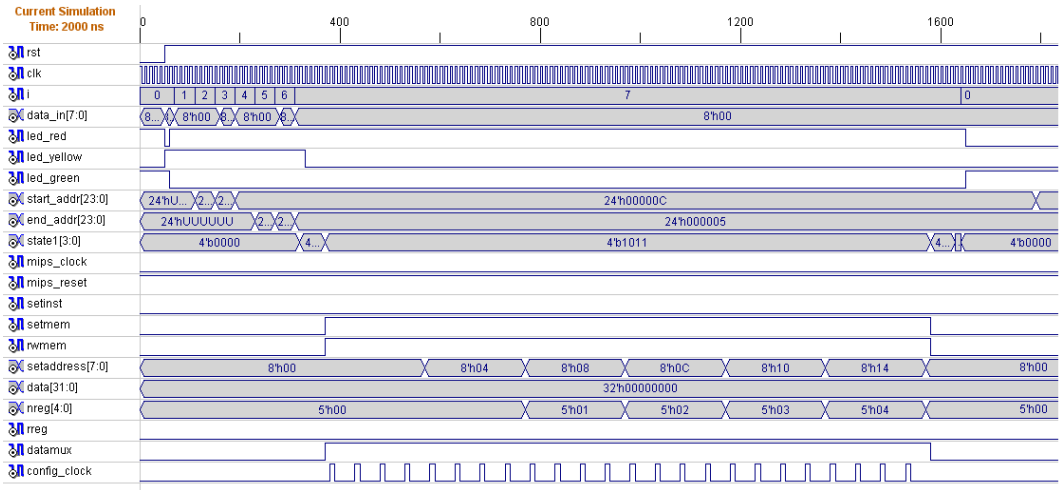


Figura 43 – Diagrama temporal dos sinais gerados numa operação de MIPS READ DATA MEMORY

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ DATA MEMORY com os parâmetros PORT=12 e Nº Words = 5;

MIPS READ DATA MEMORY (Safe mode)

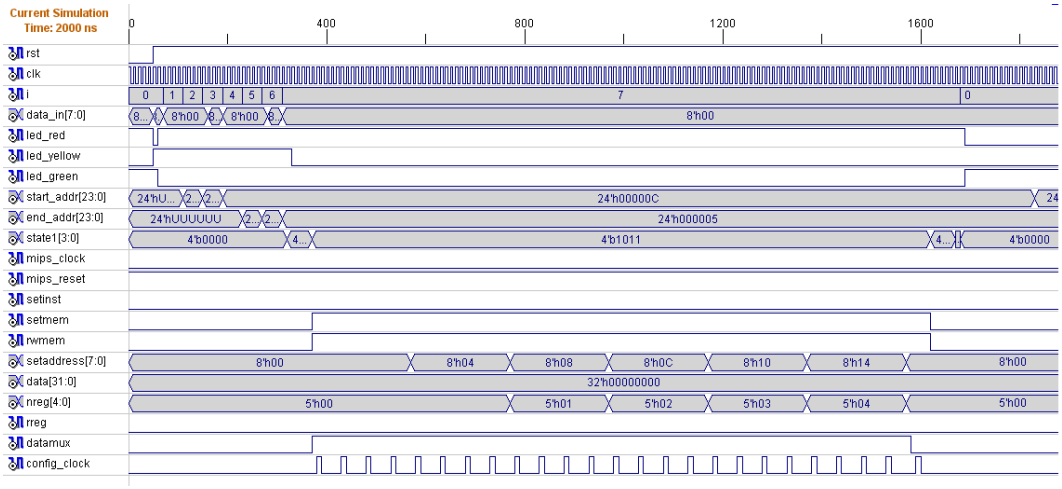


Figura 44 – Diagrama temporal dos sinais gerados numa operação de MIPS READ DATA MEMORY (Safe Mode)

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ DATA MEMORY (SAFE) com os parâmetros PORT=12 e Nº Words = 5;

MIPS SET INSTRUCTION MEMORY

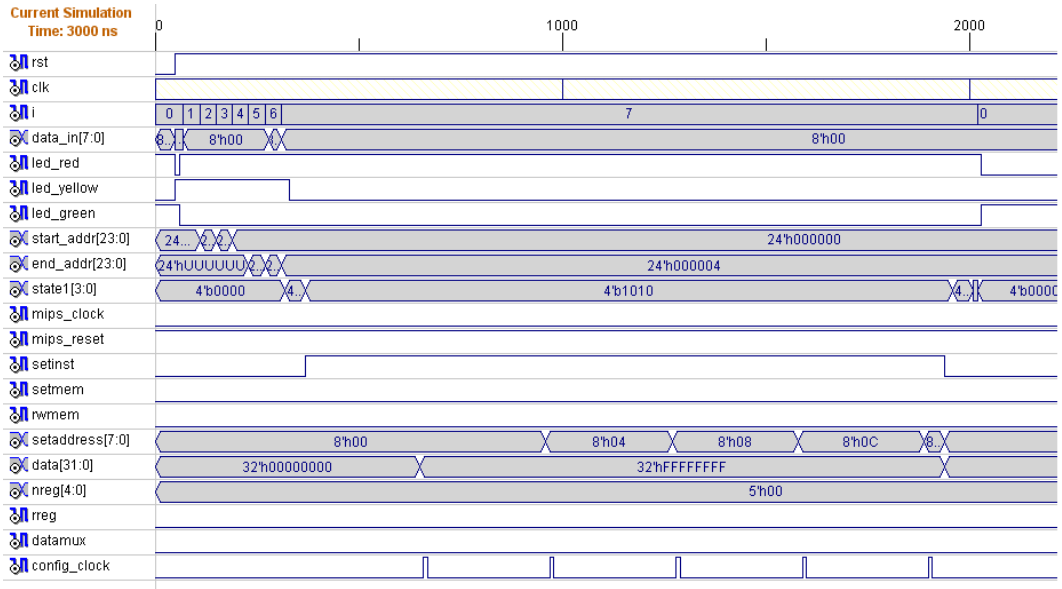


Figura 45 – Diagrama temporal dos sinais gerados numa operação de MIPS SET INSTRUCTION MEMORY

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS SET INSTRUCTION MEMORY com os parâmetros Nº Words = 4;

MIPS SET DATA MEMORY

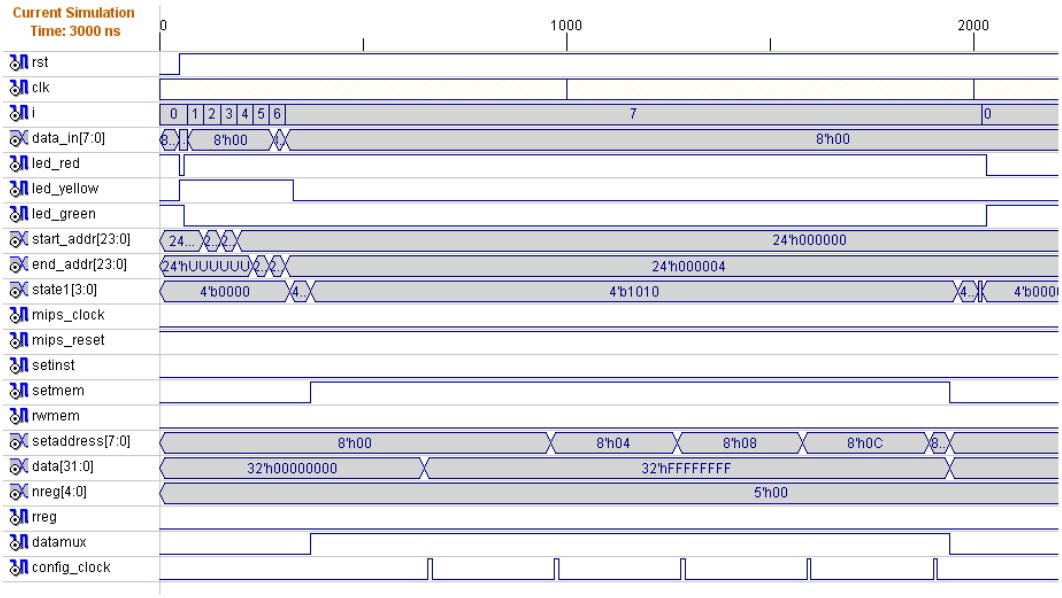


Figura 46 – Diagrama temporal dos sinais gerados numa operação de MIPS SET DATA MEMORY

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS SET DATA MEMORY com os parâmetros Nº Words = 4;

MIPS ERASE INSTRUCTION MEMORY

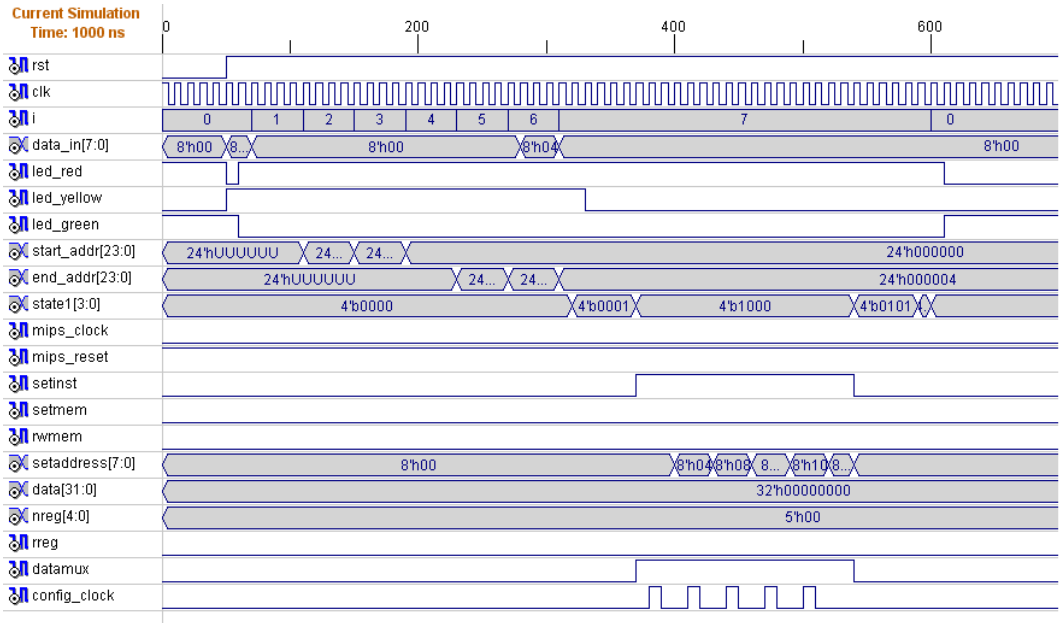


Figura 47 – Diagrama temporal dos sinais gerados numa operação de MIPS ERASE INSTRUCTION MEMORY

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS ERASE INSTUCTION MEMORY com os parâmetros Nº Words = 4;

MIPS ERASE DATA MEMORY

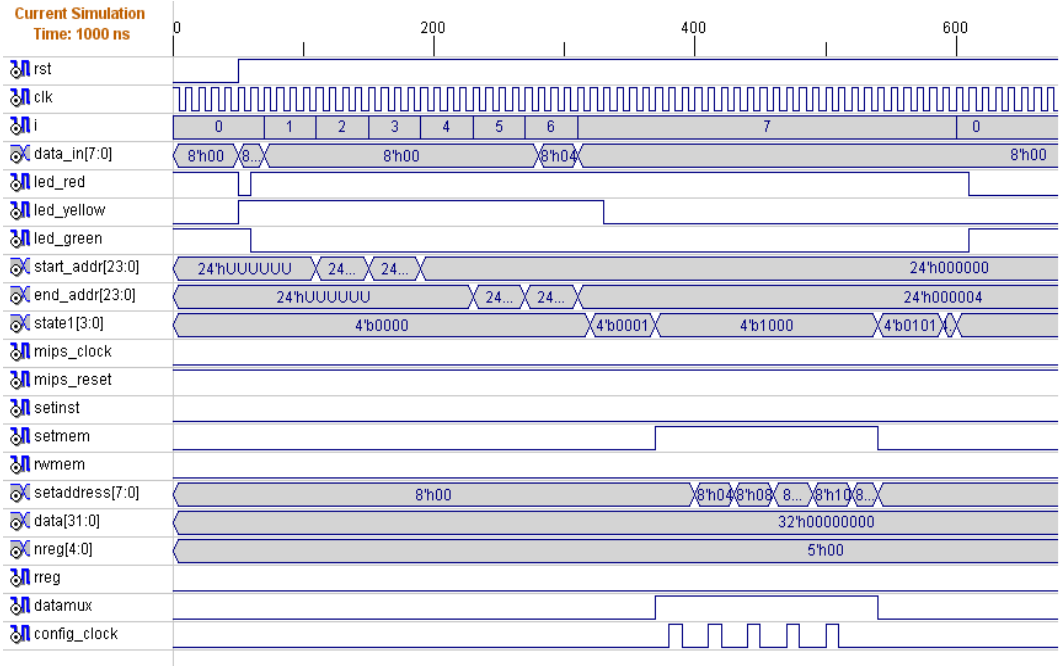


Figura 48 – Diagrama temporal dos sinais gerados numa operação de MIPS ERASE DATA MEMORY

Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS ERASE DATA MEMORY com os parâmetros Nº Words = 4;

MIPS GET DATA

Não é apresentado o diagrama temporal da evolução dos sinais gerados para a operação MIPS GET DATA, visto que é uma operação interna em que unicamente lê os valores de todos sinais ligados às portas dedicadas (PORT_*n* de 32 bits cada) e guarda-os na memória flash para posterior leitura. Nesta operação não são gerados sinais que influenciem o normal funcionamento do processador.

Da ferramenta de síntese obtiveram-se as Tabela 12 e 13 correspondentes aos valores de utilização dos recursos da FPGA Spartan 3.

MIPS32_SINGLE_CYCLE_BASIC Project Status			
Project File:	MIPS32_SINGLE_CYCLE_BASIC.ise	**Current State:**	Programming File Generated
Module Name:	project_mips	• **Errors:**	No Errors
Target Device:	xc3s400-4pq208	• **Warnings:**	27 Warnings (27 new, 0 filtered)
Product Version:	ISE 9.2.04i	• **Updated:**	sáb 6. Set 15:38:57 2008

Tabela 12 - Tabela sumária do estado do projecto
Protocolo Comunicação + MIPS32 SINGLE CYCLE

Os 27 avisos gerados são correspondentes a sinais propositadamente não ligados, situações de sinais identificados como sinais de relógio mas não o serem puramente (por exemplo: MIPS_clk), e outras informações relativas à tecnologia usada e optimizações efectuadas pela ferramenta ISE da Xilinx.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	566	7,168	7%
Number used as Flip Flops	502		
Number used as Latches	64		
Number of 4 input LUTs	2,103	7,168	29%
Logic Distribution			
Number of occupied Slices	1,251	3,584	34%
Number of Slices containing only related logic	1,251	1,251	100%
Number of Slices containing unrelated logic	0	1,251	0%
Total Number of 4 input LUTs	2,234	7,168	31%
Number used as logic	2,103		
Number used as a route-thru	131		
Number of bonded IOBs	53	141	37%
IOB Flip Flops	18		
Number of Block RAMs	4	16	25%
Number of GCLKs	5	8	62%
Number of DCMs	1	4	25%

Tabela 13 - Tabela sumária da utilização dos recursos da FPGA Spartan3 na fase de implementação do projecto Protocolo Comunicação + MIPS32 Single Cycle

Interface Gráfica

O desenvolvimento da interface gráfica foi feito de forma modular e progressiva, fruto de sucessivas interacções de planificação, modelação e construção de código que produziram múltiplas evoluções dos vários objectos e estruturas que constituem o software iCmips 1.0 de modo a contemplarem os objectivos propostos para a ferramenta.

Inicialmente foi desenvolvido um conjunto de classes que encapsulassem as estruturas e procedimentos pretendidos, como por exemplo as classes relativas à comunicação USB, armazenamento e manipulação de dados e configuração e adaptação do software ao processador implementado. Estas classes foram individualmente testadas garantido que cada uma delas efectuava com sucesso o seu objectivo.

Os primeiros testes consistiram apenas em comunicação com a placa DETIUA-S3 para geração de ciclos de relógio e aquisição de valores. Seguidamente, subindo a exigência, leitura de registos, leitura e escrita nas memórias, e por fim a edição e processamento de programas assembly.

De seguida apresento algumas imagens da execução de um programa escrito no editor de texto, processado e convertido para código máquina e por fim transmitido e executado no processador MIPS32 Single Cycle desenvolvido.

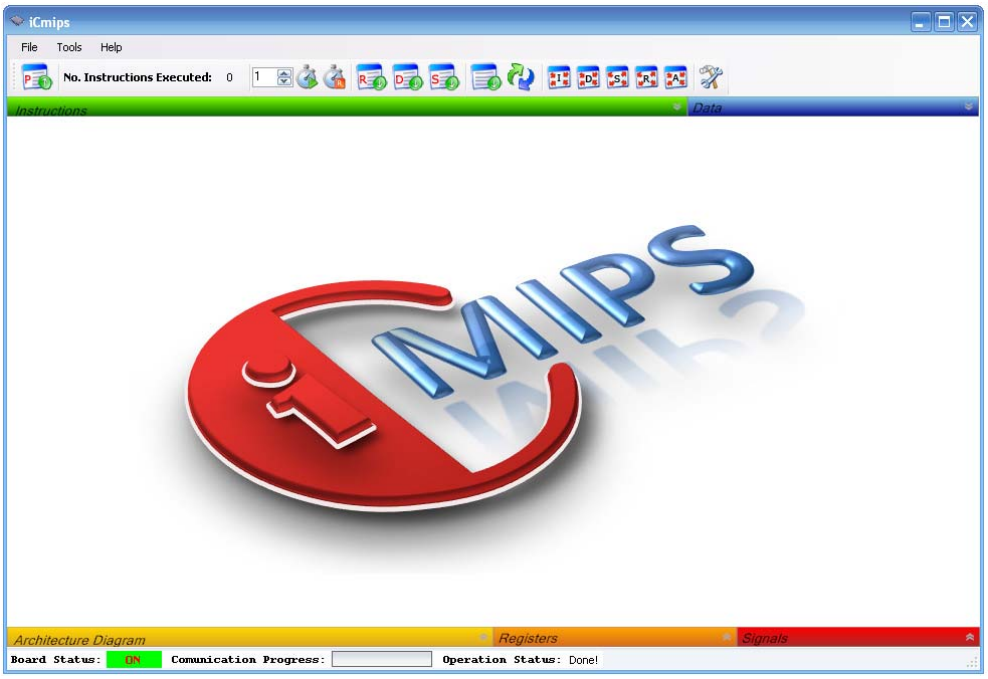


Figura 49 – Janela inicial do iCmips 1.0 com subjanelas minimizadas

As subjanelas possuem propriedades de dimensão automática ao tamanho da janela mãe, dimensionamento manual e de expandir ou minimizar

```
1  # PROGRAMA EXEMPLO iCmips
2  #
3  #     Efectua a soma de um array de 4 valores: 1+2+3+4=10
4  #
5  #
6        .data
7  ini:  .word 4,0x10,0,8
8  array:.word 1,2,0x3,0x4
9  res:  .word 0x20          # valor a ser substituido no fim do programa com o resultado
10       .align 2
11
12       .text
13       .globl main
14
15 main: nop
16       lw   $1, 0($0)     #incremento de deslocacao no array -- r1=0x4
17       lw   $3, 4($0)     #inicio do array
18       lw   $4, 8($0)     #inicializacao do somador
19       lw   $5,0x20($0)   #index seguinte ao fim do array -- r5=0x20
20       lw   $7,0x10($0)   #criterio de beq
21 loop: lw   $2, 0($3)
22       add  $4,$4,$2
23       add  $3,$3,$1      # 4xLOOP
24       slt  $6,$3,$5
25       beq  $6,$7,loop    #( -5 instrucoes )
26       sw   $4,0x24($0)
27       lw   $8,0x24($0)   #r8=0xA  - confirmacao se o LW funciona e o resultado esta correcto
```

Figura 50 – Código assembly de testes
Código assembly usado para testar o processador
MIPS32 Single Cycle e o programa iCmips 1.0

O editor de texto permite escrever o programa assembly da Figura 50 e posteriormente analisá-lo e convertê-lo no código máquina. Na Figura 51 pode-se observar a mensagem de erro dada pelo editor de texto após a tentativa do processamento do programa assembly.

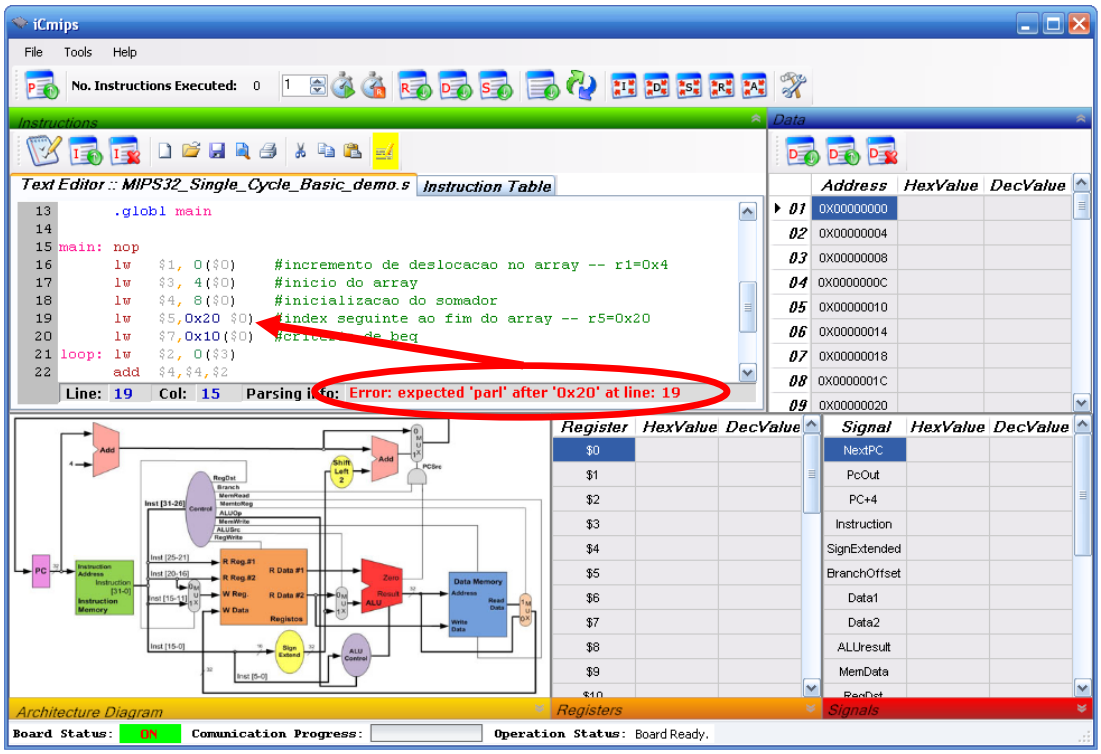


Figura 51 – Escrevendo um programa no editor de texto do iCmips 1.0

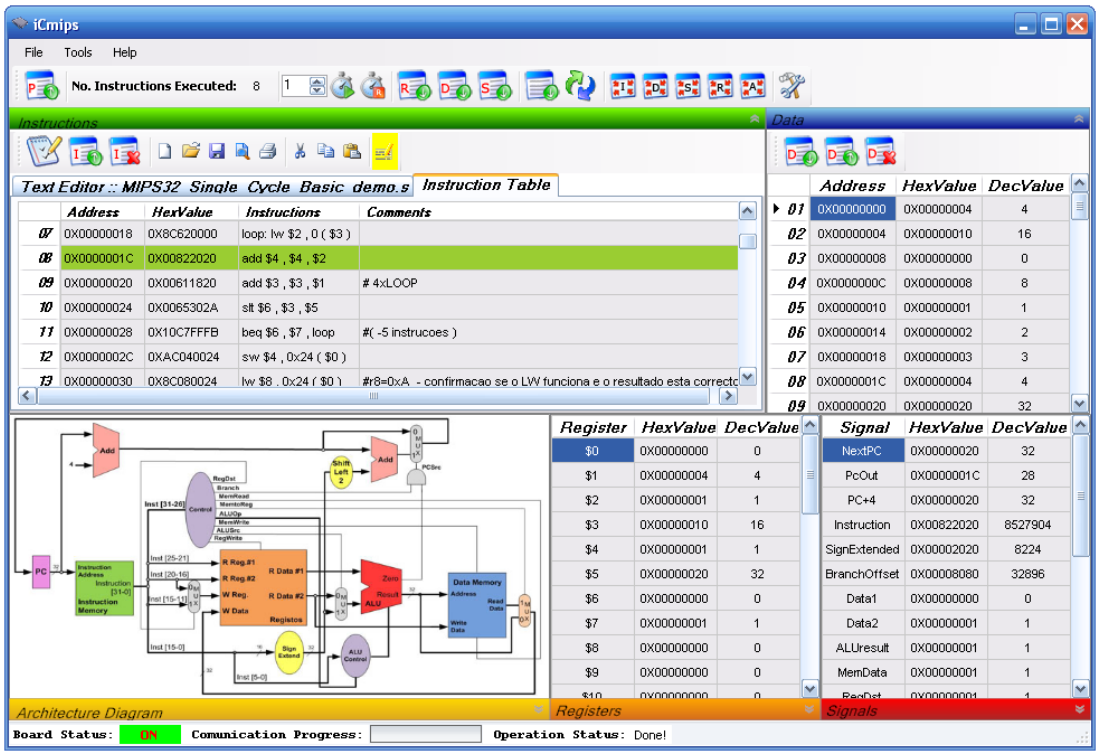


Figura 52 – Janela do iCmips durante a execução de instruções no processador

Na Figura 52 pode-se observar a validade dos resultados da execução do processador MIPS32 Single Cycle até à oitava instrução do programa apresentado na Figura 50.

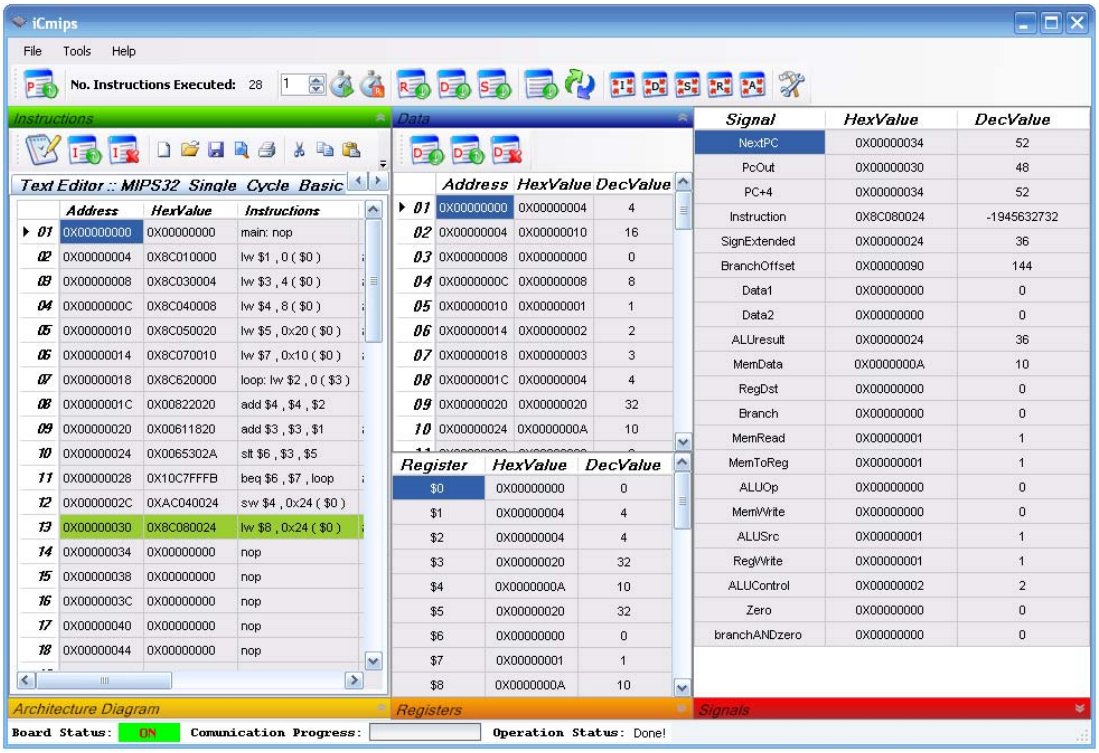


Figura 53 – Valores obtidos após a execução completa do código assembly

O iCmips 1.0 é configurado pelo utilizador para analisar um conjunto de sinais do processador ligados às portas do protocolo de comunicação e apresenta os valores lidos nas tabelas das subjanelas. Na Figura 53 pode-se observar os valores finais dos sinais gerados pelo processador MIPS32 Single Cycle no fim da execução do programa assembly apresentado na Figura 50, assim como uma nova disposição das subjanelas para melhor aproveitamento do espaço do ecrã para visualização de informação.

Apesar do funcionamento correcto da aplicação e de todos os seus módulos aos testes efectuados, o grande teste será quando chegar à sala de aula. A utilização intensiva por parte dos alunos irá contribuir à detecção de alguns possíveis erros e incoerências às quais serão prontamente corrigidas de modo a garantir a consistência da ferramenta desenvolvida.

No entanto, é sensato argumentar que a sua implementação ainda possui algumas anomalias. Refiro-me nomeadamente à instabilidade gerada pelo protocolo de comunicação que não possui um controlo de fluxo de dados, o que por vezes resulta num impasse da aplicação, sendo necessário efectuar uma nova sincronização com a placa (consiste em efectuar um reset ao protocolo de comunicação de modo a que este volte a tomar o estado inicial) para seja possível executar novas operações com a mesma.

Outra funcionalidade que ainda não se encontra totalmente bem implementada é o processamento de programas assembly. Esta funcionalidade poderia ser implementada de forma mais eficiente fazendo uso de expressões regulares em vez das verificações de condições semânticas e lexicais à linguagem assembly. O melhoramento desta funcionalidade produzirá a possibilidade de verificação de um conjunto mais vasto de situações de erro.

A ferramenta foi ainda sujeita a um conjunto de testes de execução em vários ambientes Microsoft Windows, tendo passado com sucesso no Microsoft Windows 2000, XP e Vista. Do mesmo modo, várias versões do .NET Framework foram testadas de modo a garantir o normal funcionamento da aplicação. Obteve-se sucesso em qualquer versão .NET Framework igual ou superior à 2.0.

Capítulo 5

Trabalho futuro

Os vários componentes desenvolvidos e descritos neste documento podem ser optimizados, melhorados ou redesenhados com o intuito de obter melhores resultados com os mesmos. De seguida apresento algumas das possíveis abordagens ou novos objectivos a serem implementados:

- A revisão do protocolo de comunicação será uma das primeiras coisas que deve ser contemplada, para que no futuro, a comunicação entre o computador e a placa DETIUA-S3 possua uma forma de controlo de fluxo de dados;
- A criação e integração de novos processadores MIPS, como o processador MIPS multicycle ou pipeline, possibilitará uma valorização ao projecto desenvolvido assim como novas oportunidades de ensino. O desenvolvimento de novos processadores podem até resultar da elaboração de trabalhos práticos realizados pelos alunos nas aulas laboratoriais;
- O mecanismo de multiplexers para acesso às memórias do processador e banco de registos, deve ser melhorado e simplificado. Uma das soluções possíveis é a implementação de memórias dual-port, no entanto é necessário efectuar ajustes para que as arquitecturas implementadas contemplem esse aspecto. Outra solução seria a implementação de múltiplos sinais de relógio e o acesso e execução do processador ser efectuado por sinais de controlo.
- A ferramenta de software iCmips 1.0 deverá continuar a evoluir de modo a satisfazer novas necessidades. Por exemplo:
    - A capacidade de num único ficheiro guardar toda a informação de um projecto (código VHDL, diagrama da arquitectura, programas assembly de teste e configurações da aplicação iCmips) seria uma mais-valia para a aplicação possibilitando uma melhor partilha de projectos entre alunos e professores;
    - A reformulação do assembler que analisa e processa os programas assembly produzidos no editor de texto deverá ser uma das prioridades para optimização do desempenho da aplicação e melhorar a capacidade da ferramenta processar programas assembly mais complexos face, como por exemplo, o desenvolvimento de arquitecturas com co-processadores de virgula flutuante;

- A implementação de um sistema de logs que guarde a informação gerada pelos vários componentes do processador seria interessante e útil;
- A integração na ferramenta de um simulador de execução de programas assembly, podendo assim o aluno comparar os valores obtidos no seu processador com os resultados da ferramenta de simulação;
- A criação de um mecanismo de testes automáticos ao processador, como por exemplo a possibilidade de gerar scripts de teste que no final da sua execução compara os valores gerados no processador com os esperados;
- Todas as mensagens e menus da ferramenta de software iCmips encontram-se na língua inglesa. A criação de um pacote multi-linguístico para a ferramenta possibilitará maior facilidade de utilização a alunos oriundos de outros países, efectuando Erasmus na Universidade de Aveiro.

A estrutura base da aplicação iCmips 1.0 foi pensada e modelada de forma a facilitar a implementação de alguns dos processos descritos anteriormente. No entanto como em todos os projectos, o desenvolvimento deverá ter em conta os custos e vantagens que obtêm da realização das optimizações sugeridas. Só a aplicação da ferramenta nas aulas laboratoriais permitirá avaliar os processos desenvolvidos neste projecto e futuras necessidades.

Conclusões

A implementação do processador de arquitectura MIPS, o protocolo de comunicação e o software iCmips proporcionam uma nova forma de ensino de sistemas reconfiguravéis, introdução a FPGAs e arquitectura de computadores.

Com os produtos desenvolvidos estou convicto que quando aplicado ao ensino, o professor poderá atingir resultados melhores e mais cativantes no ensino, proporcionando novos e diversificados desafios didácticos aos alunos.

O processador utilizado, sendo já conhecido dos alunos, possibilita uma aprendizagem e compreensão muito mais rápida e profunda da implementação dos componentes do processador MIPS32 Single Cycle, podendo ser melhor aproveitado o pouco tempo que um semestre possui para leccionar uma área científica tão vasta como sistemas reconfiguráveis e arquitectura de computadores.

Não foi intuito deste projecto a criação de um processador que contempla-se um elevado nível de desempenho. Pretendeu-se pelo contrário, criar um processador que permitisse executar de forma controlada e em tempo real conjuntos de instruções assembly possibilitando a visualização dos valores gerados pelo mesmo.

O protocolo de comunicação permite, além de ser usado para efeitos de comunicação com o processador desenvolvido, efectuar a comunicação com outros projectos a serem desenvolvidos, dando ao utilizador a possibilidade de acesso fácil aos valores dos sinais gerados dentro da FPGA como resultado da implementação do seu projecto para depuração de erros e validação dos objectivos pretendidos.

Com a criação do software iCmips 1.0, acredito ter desenvolvido uma ferramenta única capaz de proporcionar a todos os alunos e docentes do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro novas fronteiras ao nível de ensino e aprendizagem. A ferramenta reúne características diversas que possibilitam a análise e estudo da arquitectura MIPS, novas implementações de processadores, o ensino da linguagem assembly e mesmo a possibilidade de observar a execução física de um programa assembly num processador fisicamente implementado ao contrário do que é habitual acontecer em disciplinas de Arquitectura de Computadores a execução dos programas por simulação.

Os objectivos iniciais definidos para o desenvolvimento de um processador em VHDL para ensino foram atingidos com sucesso, podendo mesmo afirmar que foram excedidos. As ferramentas desenvolvidas encontram-se ainda em fase de análise e teste, sendo agora a sua aplicação prática a chave para futuros desenvolvimentos, optimizações e melhorias.

Referências

[1] William N. Joy, Co-Founder, and Vice President for Research on Sun Microsystems, Inc.
[Online Janeiro de 2008]:
http://www.cs.washington.edu/homes/lazowska/cra/risc.html

[2] Informações genéricas encontradas em Wikipédia:
[Online Setembro 2008]:
http://en.wikipedia.org/wiki/MIPS_architecture

[3] Informações genéricas sobre MIPS:
[Online Setembro 2008]: http://www.mips.com/

[4] "VHDL : a logic synthesis approach", Chapman & Hall, cop. 1997, first edition

[5] Skahill, Kevin," VHDL for programmable logic", Addison-Wesley, cop. 1996. XIV

[6] Informações de www.Acellera.org:
[Online Junho 2008]:
http://www.accellera.org/pressroom/2008/Accellera_VHDL_4_0_022008.pdf

[7] P.J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers, Inc.

[8] P.J. Ashenden, "The Student's Guide to VHDL", Morgan Kaufmann Publishers, Inc.

[9] Informações genéricas em Xilinx Inc.
[Online Sembro 2008]: http://www.xilinx.com

[10] Xilinx Inc. (2004), Spartan-3 FPGA Family. [Online]:
[Online Novembro 2007]:
http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[11] Chu, Pong P., " FPGA prototyping by VHDL examples : Xilinx Spartan - 3 version", Wiley-Interscience, cop. 2008.

[12] Informações em www.mitrionics.com:
[Online Setembro 2008]:
http://www.mitrionics.com/?page=products_virtual_processor

[13] Cardoso, Fabbryccio e Fernandes,Marcelo , “FPGA e Fluxo projecto”, Universidade Estatual de Campinas, Departamento de Comunicações, Brasil
[Online Agosto 2008]:
www.decom.fee.unicamp.br/~cardoso/ie344b/Introducao_FPGA_Fluxo_de_Projeto.pdf

[14] Manual de Utilizador da Placa DETIUA-S3, Versão 1.0 por Manuel Almeida
[Online Fevereiro 2008]:
www.ieeta.pt/~skl/Research/Projects/Manual_Utilizador_pt.pdf

[15] Manual de Utilização da aplicação PBM
[Online Outubro 2007]:
www.ieeta.pt/~skl/Research/Projects/PBM_User_Manual_PT.pdf

[16] Almeida, Manuel, “Métodos e Ferramentas para Reconfiguração de FPGAs Remotamente”, dissertação apresentada à Universidade de Aveiro, Departamento de Electrónica, Comunicações e Informática, 2008

[17] MacDonald, Brian , “Learning C# 2005”, 2nd ed. Sebastopol,O'Reilly, cop. 2006

[18] Sharp, John, “Microsoft visual C#. NET step by step”, Microsoft Press, cop. 2003

[19] Liberty, Jesse, “Programming C#”, 4th ed. Beijing : O'Reilly, 2005

[20] Diab, H, and I. Demashkieh, “A reconfigurable microprocessor teaching tool”, IEEE Proceedings, vol. 137, Pt. A, No. 5, September 1990

[21] Salcic, Zoran and Asim Smailagic, Digital Systems Design and Prototyping Using Field Programmable Logic, pp. 203-242, Norwell, 1997.

[22] D. Patterson and D. Hennessy, “Computer Organization and Designs: Hardware/Software Interface”, 3rd Ed., Morgan-Kaufmann, 2004

[23] Holland, Mark, “Harnessing FPGAs for Computer Architecture Education”, Master of Science in Electrical Engineering Thesis, University of Washington, 2002

[24] O’Cearuil, Diarmaid, “Design of a Teaching Instruction Set Processor in VHDL”, Final Year Project in Computer Science, University of Dublin, May 2005

[25] Sugawara, Yutaka and Hiraki, Kei, “A Computer Architecture Education Curriculum Through the Design and Implementation of Original Processors using FPGAs”, Department of Computer Science, University of Tokyo, 2004

[26] Nestor, “Teaching computer organization with HDLs: an incremental approach”, IEEE International Conference on Volume, Issue, 12-14 June 2005 Page(s): 77 – 78

[27] Hyde, Daniel C., “Teaching Design in a Computer Architecture Course”, IEEE Computer Society, Vol. 20 , Issue 3 , Pages: 23 - 28, May 2000

[28] G.M. Brown and N. Vrana, "A Computer Architecture Laboratory Course Using Programmable Logic," IEEE Trans. on Education, Vol. 38, No. 2, May 1995, pp. 118-125.

[29] Reaz M.B.I, Islam M.S., Sulaiman, M.S. , “A single clock cycle MIPS RISC processor design using VHDL”, Semiconductor Electronics, pp 199 – 203, Dec. 2002

[30] Material e trabalho desenvolvido no âmbito da disciplina Modelação e Síntese de Processadores 2007/08 [Online Novembro 2007]:
- WIKI de VHDL desenvolvido pelos alunos: http://wsl.cemed.ua.pt/vhdl/index.php
- The XST User Guide: http://wsl.cemed.ua.pt/moodle/file.php/4/docs/XSTUserGuide.pdf

[31] Tutoriais online disponíveis no âmbito da disciplina Sistemas Digitais Reconfiguráveis
[Online Setembro 2007]: http://www.ieeta.pt/~skl/SDR_V.html

[32] Material didáctico da disciplina Arquitectura de Computadores I e II 2006/2007

[33] Manual da placa Digilent Inc, Spartan-3 Starter Board online:
[Online Novembro 2007]:
http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf

[34] Pc SPIM Homepage
[Online Março 2008]: http://pages.cs.wisc.edu/~larus/spim.html

[35] Stefan Bocuti, código fonte disponível em www.codeproject.com
[Online Maio 2008]:
http://www.codeproject.com/KB/miscctrl/extendedpanel.aspx

[36] Agus Candra, código fonte disponivel em www.codeproject.com
[Online Junho 2008]:
http://www.codeproject.com/KB/miscctrl/TextEditorC_.aspx

[37] ISE Xilinx 9.2i Homepage
[Online Janeiro 2008]:
http://www.xilinx.com/ise/logic_design_prod/foundation.htm

[38] Miller, Karen “A programmer's view of computer architecture with assembly language examples from the MIPS RISC architecture”, Saunders College Publishing, cop. 1993

[39] A single clock cycle MIPS RISC processor design using VHDL Reaz M.B.I, Islam M.S., Sulaiman, M.S., Semiconductor Electronics, Dec. 2002, pp 199 – 203.

[40] MIPS Instruction Reference
[Online Novembro 2007]:
http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[41] I.Skliarova, A.B. Ferrari, “Projecto e implementação de um subconjunto da arquitectura MIPS16 com base em FPGA XC4010XL”, electrónica e Telecomunicações, v.2, nº6, Setembro 1999, pp.724-732

[42] Sweetman, Dominic, “See MIPS run ”, Morgan Kaufmann, cop. 1999

[43] Scott Hauck and André DeHon , “Reconfigurable computing : the theory and practice of FPGA-based computation”, Elsevier, cop. 2008.

Anexos

1. Artigo:

Especificação, síntese e implementação em VHDL de um processador MIPS Single Cycle Simplificado

2. Artigo:

A Tiny Multi-thread RISC MIPS Processor using VHDL

3. Tutorial de configuração da placa DETIUA-S3

4. Manual de Utilização do iCmips 1.0

Anexos

1. Artigo:

Especificação, síntese e implementação em VHDL de um processador MIPS Single Cycle Simplificado

Especificação, síntese e implementação em VHDL de um processador MIPS Single Cycle Simplificado

Bernardo Silva

***Resumo* – Este artigo descreve a implementação de circuitos reconfiguráveis que simulem um sub-conjunto da arquitectura MIPS RISC Single Cycle. O processador MIPS pode ser descomposto em cinco fases funcionais: *Instruction Fetch*, *Instruction Decode*, *Execution*, *Data Memory* e *Write Back*. A unidade de controlo opera sobre todas estas fases gerindo as operações a executar em cada uma delas. Todos os componentes constituintes desta arquitectura foram especificados em VHDL, linguagem de descrição de hardware, o que proporciona fazer o paralelismo entre descrição comportamental de hardware e implementação de circuitos digitais. Foram criados cenários de simulação de modo a efectuar a análise da funcionalidade, tempos de execução e desempenho da arquitectura implementada. Futuramente este projecto terá uma interface gráfica que permitirá uma visualização em tempo real dos valores dos sinais que constituem a arquitectura do processador desenvolvido. O projecto em desenvolvimento poderá ser usado no âmbito das disciplinas Computação Reconfigurável (4º ano de MIECT), Sistemas Digitais Reconfiguráveis (opção de 5º ano, MIEET) e Modelação e Síntese de Processadores (opção de 5º ano, MIECT/MIEET).**

***Abstract* – This paper describes an implementation of reconfigurable circuits which emulate an instruction subset of a simplified MIPS RISC Single Cycle processor. The MIPS processor can be decomposed in five functional stages: Instruction Fetch, Instruction Decode, Execution, Data Memory, and Write Back. The Control Unit operates in all of these stages, managing the way each operation should be executed. All the components of the architecture were specified using VHDL, allowing to establish the parallelism between behavioral hardware description and circuit implementation. Different simulation scenarios were created to analyze the functionality of the designed system, execution times and performance. In the near future, a graphical interface is going to be developed, making it possible to visualize the values of the processor's signals in real time. The designed project can be successfully employed within Reconfigurable Computing (4th year of Computer Engineering curriculum), Reconfigurable Digital Systems (5th year, Electrical Engineering curriculum) and Processor Synthesis and Modeling (5th year, Computer/Electrical engineering curriculum) disciplines.**

I. INTRODUÇÃO

Ao longo dos anos, tem-se vindo a verificar um grande aumento de desempenho e integração nos componentes lógicos em FPGA permitindo implementar sistemas lógicos complexos num único *chip* [1]. O desenvolvimento de sistemas exigentes e complexos pode implicar a implementação de circuitos com milhões de portas lógicas que incluem memórias, interfaces rápidas e outros componentes de alto desempenho. Uma das abordagens para o projecto e implementação deste nível de complexidade é a utilização de linguagens de descrição de hardware, por exemplo VHDL (Very high speed integrated circuit Hardware Description Language). Esta permite a um projectista de sistemas desenvolver hardware de forma simples e rápida, com a grande vantagem de ser possível efectuar simulações e múltiplas implementações o que torna as fases de desenvolvimento mais rápidas e diminuem os custos que uma produção física de várias evoluções do sistema poderia ter até atingir o produto final [2].

A filosofia RISC (Reduced Instruction Set Computer) teve inícios nos anos setenta com a IBM e desde então foi desenvolvida e documentada em universidades e companhias de microprocessadores. O intuito desta filosofia é de optimizar e obter o máximo de desempenho de um processador e ao mesmo tempo simplificar o hardware de modo a conseguir custos razoáveis de produção [3]. Na arquitectura RISC o conjunto de instruções é baseado em abordagens *load/store*. Apenas o conjunto de instruções *load* e *store* permite o acesso à memória. Nenhuma das outras operações aritméticas ou de I/O interagem com a memória. A simplificação neste tipo de arquitectura resulta numa rápida descodificação de instruções e facilidade de implementação. Devido à sua robustez de desempenho, formatos simples de instruções, diversidade e suporte de produtos, o processador MIPS RISC, é neste momento um dos líderes de mercado [4].

Neste artigo, encontra uma abordagem de implementação do processador MIPS Single Cycle usando a linguagem de descrição de hardware VHDL. O objectivo deste trabalho é a demonstração de como VHDL é fiável e permite a implementação rápida de protótipos de microprocessadores. O uso de VHDL acresce de variadas vantagens para este estudo, nomeadamente como a

possibilidade de implementação a vários níveis de especificação e encadeamento das estruturas necessárias (arquitectura, registos, blocos de memória, barramentos, portas lógicas) durante todo o processo de desenvolvimento [5].

II. PRINCÍPIOS DE FUNCIONAMENTO E IMPLEMENTAÇÃO

A arquitectura do processador MIPS é baseada nos três tipos de instruções de 32 bits cada: Tipo-R, Tipo-J e Tipo-I, fazendo com que a arquitectura desenvolvida possua um *datapath* de 32 bits consistente com a dimensão das instruções (Tabela 1). No desenvolvimento deste projecto não foi implementado o Tipo-J, no entanto facilmente é acrescentada essa possibilidade.

Tipo-R						
Campo	Opcode	Rs	Rt	Rd	Shamt	Funct
Bits	31-26	25-21	20-16	15-11	10-6	5-0
Tipo-I						
Campo	Opcode	Rs	Rt	Address		
Bits	31-26	25-21	20-16	15-0		

Tabela 1. Formato dos tipos de instruções implementadas

A implementação de um processador deste tipo implica que num ciclo de relógio sejam efectuadas as fases de busca da instrução, a sua descodificação, execução ou acesso à memória e armazenamento dos dados produzidos. No entanto, como se pode verificar ao construir esta arquitectura, não é de todo possível num único ciclo de relógio efectuar todas as sincronizações necessárias. Deste modo o ciclo de relógio é dividido por dois, podendo assim usufruir de 4 flancos de sincronização distribuídos pelos vários componentes da arquitectura que os necessitam. Foi atribuído o nome de *CPUClock* ao sinal de relógio que por cada ciclo, efectua uma instrução e *MemClock* ao sinal de relógio que possui o dobro da frequência de *CPUClock*, sendo este associado às leituras e escritas das memórias (Figura 1).

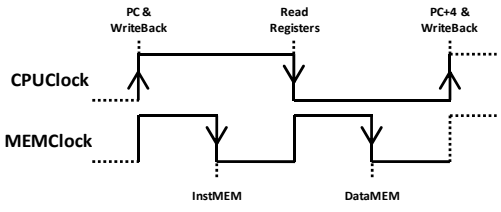


Figura 1. Diagrama representativo dos acessos efectuados nos flancos de relógio de *CPUClock* e *MEMClock*

A. Fase Instruction Fetch

A primeira fase da execução no MIPS RISC é a leitura da memória da instrução que se pretende executar (IF – *Intruction Fetch*). Esta operação de leitura é iniciada quando o *Program Counter* (PC), registo de 32 bits liberta um endereço ao bloco *Instruction Memory* no flanco ascendente do *CPUClock*. O PC é incrementado de modo a indicar o próximo endereço de acesso à memória para a instrução seguinte (*PC+4*). No bloco *Instruction Memory*, lê-se da memória a instrução apontada pelo endereço recebido. Esta operação de leitura de memória é sincronizada com o flanco descendente de *MEMClock* e apenas efectuada quando *CPUClock* possui o valor '1' (ver transição *InstMEM* na Figura 1).

No caso de instruções de salto (Branch), o endereço da instrução seguinte poderá ser substituído pelo endereço gerado pela situação de salto. Para que tal aconteça com sucesso, é necessária lógica adicional que calcule o endereço de salto. Um multiplexer de duas entradas de 32 bits selecciona qual das suas entradas deve ser colocada à saída através do resultado gerado por uma porta lógica AND que avalia se a condição de salto é satisfeita.

A memória de instruções foi implementada com base nos blocos de memória embutida BlockRAM, disponíveis em FPGAs da família Spartan-3 [6]. Esta memória foi inicializada já com instruções que possibilitem posteriormente a análise dos resultados obtidos em simulação e detecção de erros [2,7,8].

Um diagrama de bloco desta fase é visualizado na Figura 2.

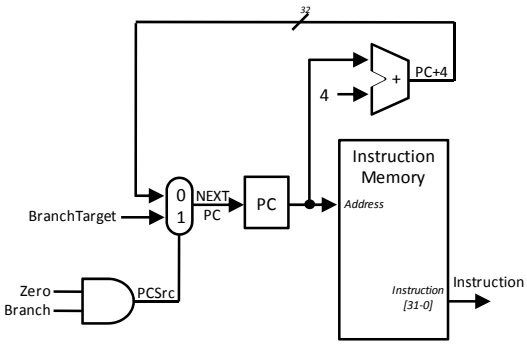


Figura 2. Diagrama da fase de *Intruction Fetch*

B. Instruction Decode

Quando a instrução é lida e colocada no barramento de saída do bloco *Instruction Memory*, os vários campos que constituem a instrução são processados. O *Opcode* da instrução é lido pela unidade de controlo do *datapath* e o campo *Funct* lido pela unidade de controlo da *ALU* (Arithmetic Logical Unit). Nesta fase a unidade de controlo (*Control Unit*) após ler o *Opcode* da instrução dispõe à sua saída todos os sinais de controlo do datapath (descrito mais detalhadamente à frente).

Os campos da instrução correspondentes aos registos que se pretendem ler/escrever são lidos pelo módulo *File Register*. Na implementação efectuada, as leituras de registos são assíncronas (podendo ser

efectuadas de forma síncrona, ver transição *ReadRegisters* na Figura 1) e a escrita de dados no registo de destino é síncrona com o flanco ascendente de *CPUClock* e apenas executada quando o sinal *RegWrite* se encontra activo.

Os campos lidos, cada um com 5 bits, endereçam registos de 32 bits que são lidos e colocados na saída do módulo. Existem 32 registos [r0-r31], todos disponíveis para guardar valores gerados pelas instruções à excepção do registo r0 que possui sempre o valor zero, norma implementada na arquitectura MIPS, visto que é um valor muitas vezes usado durante a execução de instruções.

O módulo *File Register* possui então quatro entradas: correspondentes aos endereços de dois registos de leitura e um de escrita, um porto de entrada de 32 bits dos dados a escrever e duas saídas de valores contidos nos dois registos lidos: *RegData1* e *RegData2*, ambos de 32 bits. Este módulo foi desenvolvido em VHDL de modo a que após sua síntese, os registos são implementados em LUTs (*LookUp Tables*) [3,8].

Os 16 bits menos significativos da instrução passam pelo *SignExtended*, módulo que replica o bit mais significativo de forma a gerar uma saída de 32 bits em complemento para 2. Este valor, servirá posteriormente para cálculo de endereços de salto, acesso à memória para leitura/escrita ou leitura de valores imediatos em instruções contendo valores imediatos.

O diagrama de blocos da figura 3, correspondente a esta fase, representa o descrito anteriormente.

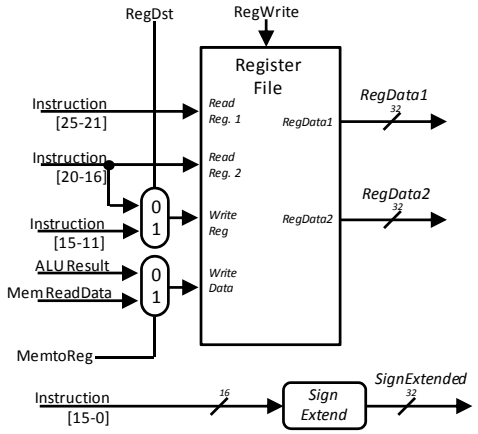


Figura 3. Diagrama de blocos da fase de *Intruction Decode*

C. Execution

Após a descodificação da instrução na fase anterior, dá-se início à fase *Execution* que produz todos os cálculos necessários. Os componentes nesta fase são a *ALU*, *ALU Control* e os blocos necessários para cálculo do endereço de salto: um somador de 32 bits e bloco combinatório de deslocamento *SLL2*. Na unidade de *ALU* efectuam-se os deslocamentos, cálculos aritméticos e lógicos e utiliza-se o somador de 32 bits para determinar o endereço relativo de salto duma instrução *branch*. Para efectuar o cálculo do endereço é necessário antes efectuar um deslocamento lógico à esquerda de dois bits do sinal *SignExtended* e depois usar esse resultado e o valor de *PC+4* como entradas do somador de 32 bits. O diagrama de blocos da figura 4 demonstra a arquitectura desta fase.

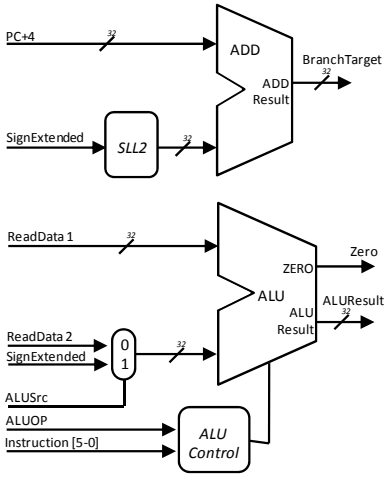


Figura 4. Diagrama de blocos da fase *Execution*

A ALU possui 3 entradas, sendo duas correspondentes aos valores dos registos no caso de instruções do Tipo-R ou o valor do primeiro registo e os 32 bits do *SignExtended* para instruções do Tipo-I. A selecção da segunda entrada é efectuada através de um multiplexer controlado pela unidade de controlo. A terceira entrada é correspondente aos bits de controlo. Estes bits são gerados pela *ALU Control* que analisa os 6 bits do campo *Funct* das instruções e os 2 bits gerados pelo *Control Unit*, decifrando assim que tipo de operação se pretende efectuar na *ALU*. A saída da *ALU Control* é um sinal de 3 bits que permitem gerar uma das seguintes combinações mostradas na Tabela 2.

Instrução	Tipo de instrução	ALU OP	Operação a executar	Campo Funct	Operação da ALU	Controlo da ALU
LW	Tipo-I	00	LOAD WORD	XXXXXX	ADD	010
SW	Tipo-I	00	STORE WORD	XXXXXX	ADD	010
BEQ	Tipo-I	01	BRANCH EQUAL	XXXXXX	SUBTRACT	110
ADD	Tipo-R	10	ADD	100000	ADD	010
SUB	Tipo-R	10	SUBTRACT	100010	SUBTRACT	110
AND	Tipo-R	10	AND	100100	AND	000
OR	Tipo-R	10	OR	100101	OR	001
SLT	Tipo-R	10	SET ON LESS THAN	101010	SET ON LESS THAN	111

Tabela 2. Sinais gerados pelo ALU Control em função do ALUOP e Funct recebidos

O uso de VHDL, fornece grandes vantagens na descrição das condições que se pretendem verificar, usando declarações *CASE-IS* ou *IF-THEN-ELSE* pode-se

mesmo quase transcrever a leitura da Tabela 2, exemplo disso é a Figura 5 que representa a implementação usada em VHDL para o módulo *ALUControl*.

```
process(Funct, ALUOp)
begin
        case ALUOp is
        when "00" => Output <= "010";
        when "01" => Output <= "110";
        when others =>
                case Funct is
                when "100100" => Output <= "000";
                when "100101" => Output <= "001";
                when "100000" => Output <= "010";
                when "100010" => Output <= "110";
                when others   => Output <= "111";
                end case;
        end case;
end process;
```

Figura 5 – Código VHDL do módulo *ALUControl* usando declarações do tipo *CASE-IS*

A *ALU* possui dois sinais de saída, em que num coloca o resultado da operação aritmética/lógica efectuada (*ALUResult*) e noutro um sinal de um bit chamado *Zero*, que fica activo quando o resultado da ALU for igual a zero. Este último sinal é ligado à porta lógica AND que recebe também um sinal do *Control Unit* denominado de *Branch* (activo numa situação de salto conforme ilustrado na Figura 2). Quando estes dois sinais estão activos a condição de salto é validada e o valor do próximo endereço de memória de instruções que deve ser lido é o apontando pela instrução de salto.

D. Data Memory

A fase *Data Memory*, lê ou escreve valores da ou para a memória. Esta fase numa situação de *Branch* não produz qualquer tipo de efeito. A *Data Memory* (Figura 6) recebe dois valores à entrada correspondentes aos dados a serem escritos (*WriteData*) e o endereço de memória ao qual se pretende aceder (*Address*). Este módulo possui ainda um porto de saída no qual coloca o valor lido da memória (*ReadData*). Dois sinais separados, *MemWrite* e *MemRead*, são usados para garantir a escrita ou a leitura da memória, sendo esta acção sincronizada com o *MEMClock* e *CPUClock* (ver transição *DataMem* na Figura 1).

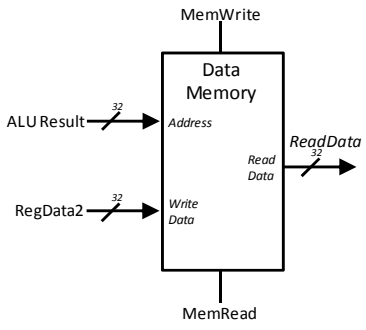


Figura 6. Diagrama do bloco da fase de *Data Memory*

Para ler da memória o sinal *MemRead* encontra-se activo e *MemWrite* desactivo. Para escrita estes sinais encontram-se invertidos, sendo por isso possível, se pretendido, uma optimização de recursos usando apenas um bit de controlo para escrita e leitura. Durante o desenvolvimento, uma pequena parcela de memória foi inicializada para efeitos de simulação e verificação de erros. A implementação deste tipo de Memória foi feita em módulos BlockRAM, usufruindo das vantagens das estruturas disponibilizadas pela FPGA. Em VHDL uma BlockRAM pode ser instanciada directamente ou usando uma sequência de instruções VHDL que a ferramenta de síntese (XST da Xilinx) consegue interpretar como sendo objectivo do projectista a implementação em BlockRAM do componente descrito. De seguida encontra-se uma parte parcial do código que demonstra a descrição em VHDL da memória de dados (Figura 7).

```
signal Memory : T_DataMem := DataMemory;

process (Clock, Enable)
begin
   if FALLING_EDGE(Clock)  then
        if MemRead = '1' and Enable = '0' then
           ReadData <= Memory(CONV_INTEGER(Address));
        elsif MemWrite = '1' and Enable = '0' then
           Memory(CONV_INTEGER(Address)) <= WriteData;
        end if;
   end if;
end process;
```

Figura 7 – Código VHDL que descreve a implementação de uma memória

E. Write Back

Nesta fase de uma instrução MIPS RISC, o resultado produzido em fases anteriores como *Execution* ou *DataMemory* pode ser escrito num registo. Um multiplexer determina qual dos resultados obtidos deve ser escrito no registo de destino. Esta operação de selecção é gerida pelo *Control Unit* que através do sinal *MemToReg* efectua o controlo sobre este multiplexer. As instruções envolvidas nesta fase são operações de Registo-Registo ou *Loadword*. Para operações Registo-Registo, o resultado da ALU é passado para escrita sendo necessário para isso o bit de selecção do multiplexer conter o valor '0' e para valores lidos da memória, encontra-se no sinal de selecção o valor activo "1", escrevendo assim no registo o conteúdo do sinal *ReadData* (pode-se visualizar o diagrama desta situação na Figura 8).

F. Control Unit

Em todas as fases do MIPS RISC, existem sinais de controlo como já referidos anteriormente. Estes, estão descritos na Tabela 3, e possuem a capacidade de alterar o funcionamento do *datapath* dependente da instrução que se pretende executar.

Fase do MIPS RISC	Sinais de controlo	Efeito dos sinais
Instruction Fetch	PCSrc	O valor do PC é substituído pelo endereço calculado pela instrução de salto ou pelo valor de PC+4.
Instruction Decode	RegDst	O número do registo de destino provém do valor dos bits 15-11 ou dos bits 20-16 da instrução.
	RegWrite	O registo de destino escolhido é escrito com o conteúdo do sinal ligado à entrada do porto WriteData.
Execution	ALUSrc	O segundo operando da ALU é os 16 bits menos significativos da instrução extendidos de sinal até 32 bits. Caso contrário é o valor do segundo registo lido (ReadData2).
Data Memory	MemWrite	O conteúdo da memória apontado pelo endereço é escrito com o valor à entrada do porto WriteData.
	MemRead	O conteúdo da memória apontado pelo endereço é lido e colocado no porto de saída (ReadData).
Write Back	MemtoReg	O valor que se encontra para escrita no registo provém da memória ou do resultado da ALU.

Tabela 3. Sinais de controlo em cada fase e seu efeito quando activos

A unidade de controlo (*Control Unit*) recebe à entrada o *Opcode* da instrução e gera na sua saída os sinais necessários para controlo de registos, memória, multiplexers e *ALUControl*.

Como só depende do *Opcode*, em cada ciclo de relógio os sinais de controlo apenas podem possuir o valor '1', '0' ou ser irrelevantes (*don't-care*) como se pode verificar na Tabela 4. No desenvolvimento em VHDL, o módulo *Control Unit* foi construído utilizando *CASE-IS* declarando de uma forma explícita todos os sinais gerados para cada *Opcode* de entrada.

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp 1	ALUOp 0
Tipo-R	1	0	0	1	0	0	0	1	0
LW	0	1	1	1	1	0	0	0	0
SW	X	1	X	0	0	1	0	0	0
BEQ	X	0	X	0	0	0	1	0	1

Tabela 4. Valores atribuídos aos sinais de controlo dependendo do tipo de instrução

Ao combinar todas as fases anteriormente descritas obtém-se o *datapath* da arquitectura pretendida para um conjunto reduzido de instruções MIPS de 32 bits (ver Figura 8).

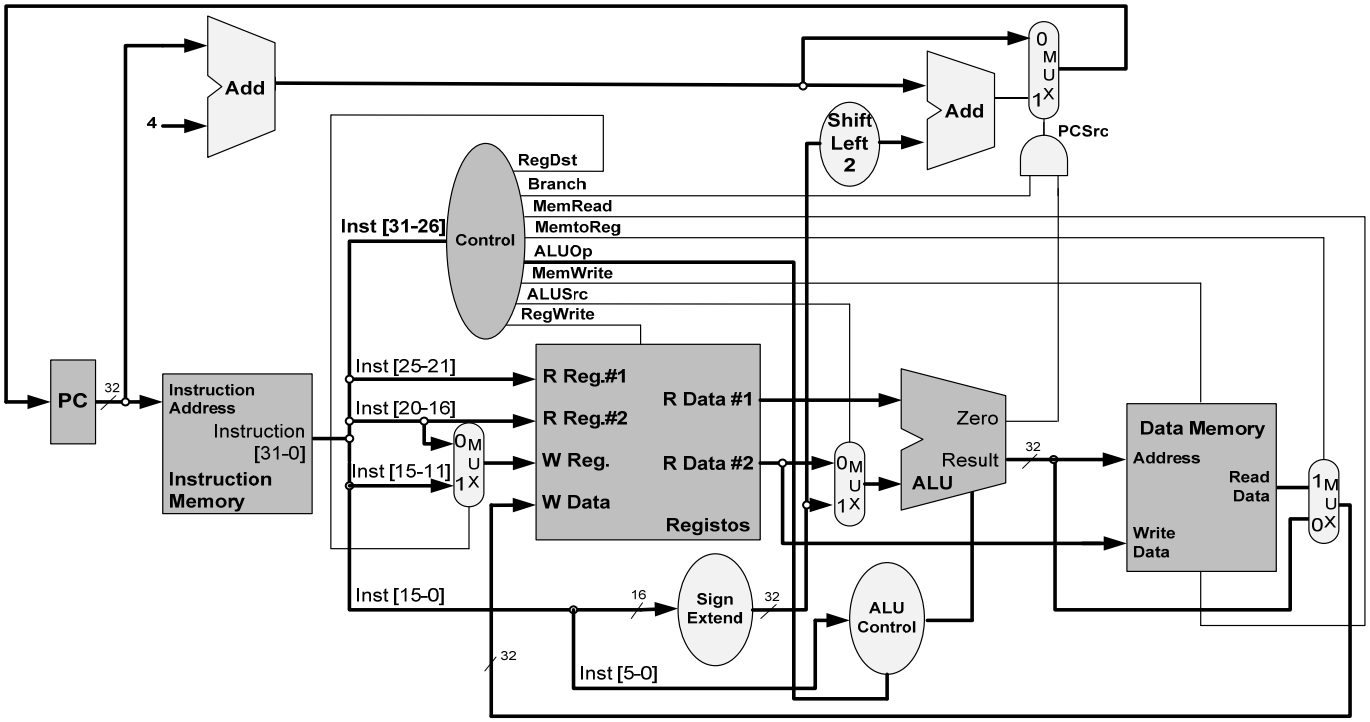


Figura 8. Arquitectura completa do MIPS RISC de 32 bits simplificado com a unidade de controlo e a fase de *Write Back* (imagem adapatada de [9])

III. SIMULAÇÃO E ANÁLISE

Para síntese e simulação deste projecto usou-se a ferramenta ISE Xilinx 9.2.04i [10]. Esta ferramenta possui um simulador integrado (ISE Simulator) no qual permitiu executar um trecho de código assembly (apresentado na Tabela 5) na arquitectura projectada e analisar os resultados obtidos com os valores esperados. Uma parte da memória de dados foi inicializada com alguns valores conforme se pode ver na Tabela 6.

Nº da Instrução	Endereço de Memória (Hex)	Instrução Assembly (valores em hexadecimal)
1.	00	lw r1, 0(r0)
2.	04	lw r3, 4(r0)
3.	08	lw r4, 8(r0)
4.	0C	lw r5,20(r0)
5.	10	lw r7,10(r0)
6.	14	Loop: lw r2, 0(r3)
7.	18	add r4,r4,r2
8.	1C	add r3,r3,r1
9.	20	slt r6,r3,r5
10.	24	beq r6,r7,Loop
11.	28	sw r4,24(r0)
12.	2C	lw r8,24(r0)

Tabela 5. Conteúdo da *Instruction Memory*

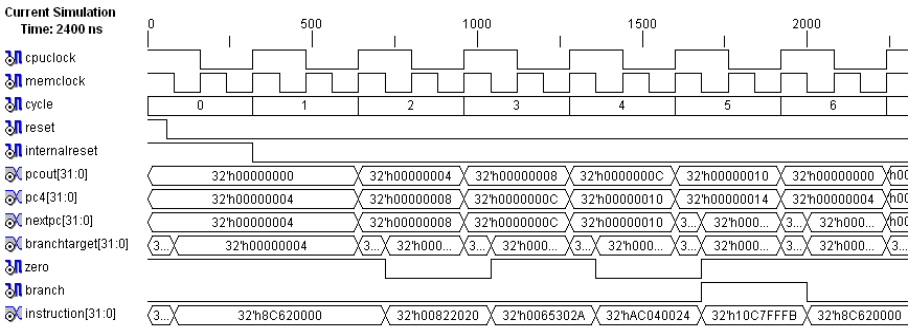
O código assembly consiste num conjunto inicial de instruções de leitura de valores da memória, seguido de um ciclo de 4 iterações que efectua o somatório dos elementos de um array de valores inteiros.

Endereço de Memória (Hex)	Conteúdo da Memória de Dados (Hex)
00	0x04
04	0x10
08	0x00
0C	0x08
10	0x01
14	0x02
18	0x03
1C	0x04
20	0x20
24	0x00

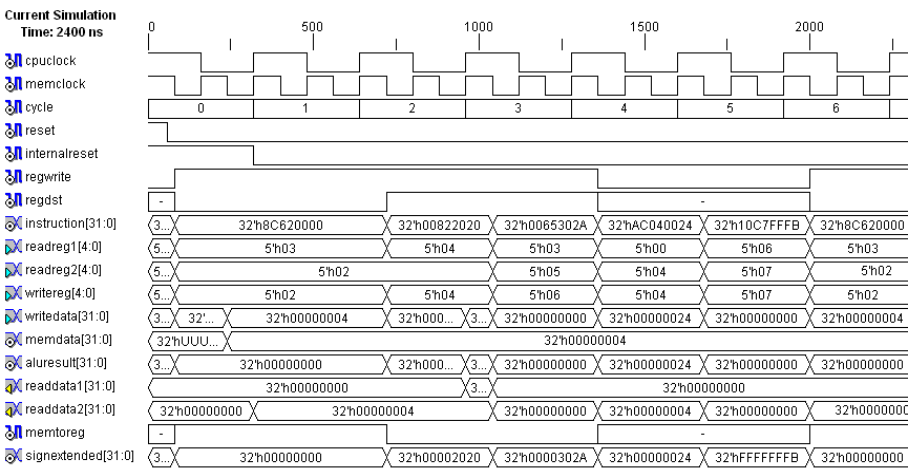
Tabela 6. Conteúdo da *DataMemory* após a inicialização

Por fim, o valor do somatório é escrito em memória, e novamente lido. Em seguida são mostrados alguns dos resultados obtidos correspondentes à simulação sequencial das instruções número 6, 7, 9, 11 e 10 (a troca das instruções 10 e 11 foi propositada para efeitos de demonstração das simulações). Deste modo pode-se verificar o PC a iniciar com o valor 0x00 na instrução 6 até ao valor 0x10 da instrução 10, voltando novamente a 0x00 devido à instrução 10 corresponder a um *branch taken*, neste subconjunto de instruções simuladas. O subconjunto de instruções escolhido permite visualizar os valores dos sinais no *datapath* para diferentes formatos de instrução.

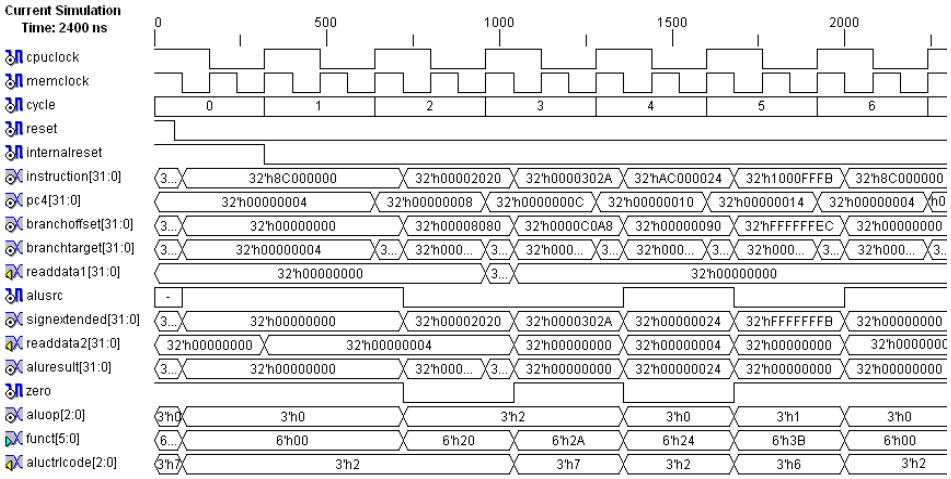
SIMULAÇÃO DA FASE INSTRUCTION FETCH:



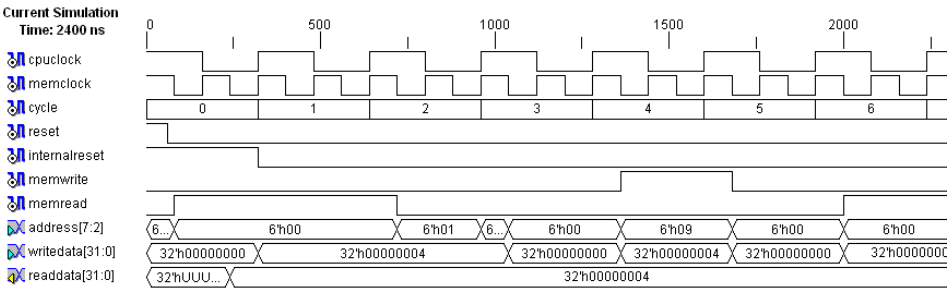
SIMULAÇÃO DA FASE INSTRUCTION DECODE:



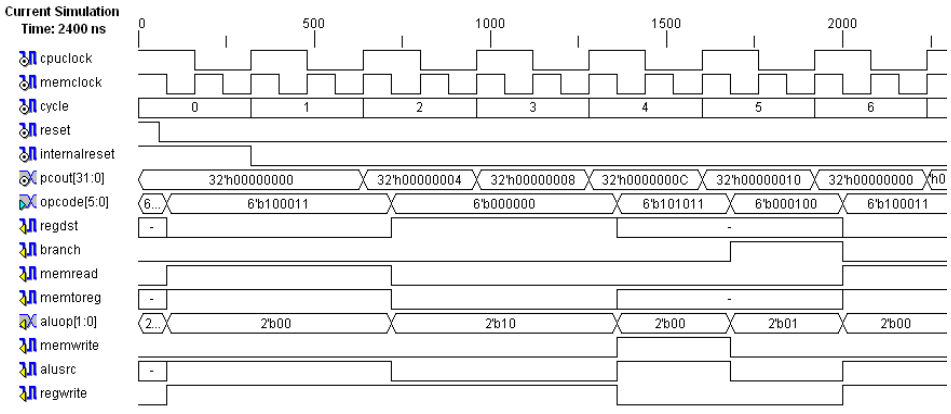
SIMULAÇÃO DA FASE EXECUTION:



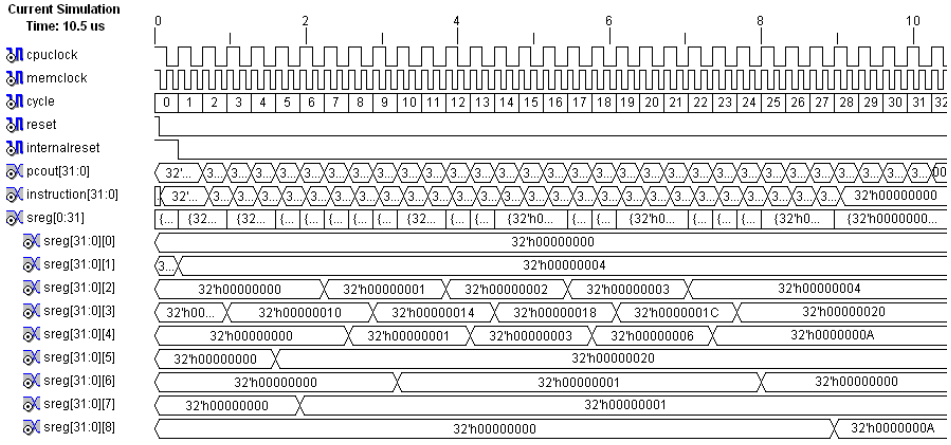
SIMULAÇÃO DA FASE DATAMEMORY E WRITEBACK:



SIMULAÇÃO DA UNIDADE DE CONTROLO:



SIMULAÇÃO DA ARQUITECTURA DESENVOLVIDA PARA TODAS AS INSTRUÇÕES DA TABELA 5:



No fim desta execução de código é possível verificar que os valores finais correspondem aos pretendidos, demonstrando assim que todos os valores foram devidamente gerados e calculados.

Foi também implementada uma versão ligeiramente alterada, no âmbito da disciplina de opção do 5º ano para MIECT e MIEET, Modelação e Síntese de Processadores, que permitiu visualizar o resultado final iluminando 8 LEDs correspondentes aos 8 bits menos significativos do resultado pretendido para o registo r8. Isto foi possível usando o bit 7 do resultado da ALU para decisão de escrita para os LEDs. Quando a '1' escreve na memória de dados e também para os LEDs da placa *Spartan 3 Starter Board* da *Digilent Inc* [10], iluminando-os.

IV. FUTURO TRABALHO

Um dos objectivos pretendidos neste projecto é que de uma forma fácil e construtiva, um aluno possa ir aprendendo VHDL. Para o efeito, será de todo interessante o aluno poder efectuar uma implementação pessoal deste tipo de arquitectura e posteriormente possuir uma ferramenta que permita visualizar os valores contidos nos sinais em cada ciclo de relógio durante a execução de cada instrução na arquitectura MIPS RISC desenvolvida. Uma interface gráfica está a ser desenvolvida em C#, de modo a tirar partido do facto de a placa DETIUA-S3 [12] já possuir uma ligação USB pela qual se efectua toda a transmissão de dados de sua configuração. Assim, será criada uma máquina de estados que implemente um protocolo de comunicação entre o módulo MIPS desenvolvido e a memória flash da placa DETIUA-S3. Utilizando este protocolo de comunicação será possível que o utilizador possua uma maior interactividade no que diz respeito a carregamento de instruções, inicialização da memória de dados, controlo de ciclo de relógio e visualização de valores da implementação do seu próprio código VHDL de uma versão MIPS. Se tais objectivos forem alcançados, quem sabe, um futuro aluno da Universidade de Aveiro poderá aprender e ter uma maior proximidade com código assembly, Arquitectura de Computadores, VHDL, síntese de processadores e circuitos digitais numa única ferramenta.

V. CONCLUSÕES

Ao longo do desenvolvimento deste projecto foram adquiridos diversificados conhecimentos em VHDL. Estes conhecimentos contribuíram para sucessivas optimizações de alguns dos módulos. Em título de referência, a utilização de um pacote em VHDL para declarar as dimensões do barramento do *datapath*, tamanho das memórias e inicializações das mesmas e mesmo alguns tipos de dados. Desta forma o MIPS implementado possui a característica de possuir um código VHDL simples, legível e (apesar de limitado) parametrizável.

Este tipo de projecto veio também a proporcionar a aplicação de vários conhecimentos adquiridos no âmbito de múltiplas disciplinas do curso de Mestrado Integrado em Engenharia de Computadores e Telemática.

Por simulação após a verificação dos resultados obtidos corresponderem aos pretendidos, e utilização de outras sequências de instruções, pequenos programas em assembly, pode-se afirmar que o processador MIPS Single Cycle simplificado, implementado em VHDL foi desenvolvido e testado à frequência de relógio de 50 MHz na placa *Spartan-3 Starter Board* com sucesso.

REFERÊNCIAS

[1] Linley Gwennap , FPGA Evolution, online: http://www.linleygroup.com/columns/nikkei1006.html

[2] Material e trabalho desenvolvido no âmbito da disciplina Modelação e Síntese de Processadores 2007/08:
- WIKI de VHDL desenvolvido pelos alunos: http://wsl.cemed.ua.pt/vhdl/index.php
- The *XST User Guide:* http://wsl.cemed.ua.pt/moodle/file.php/4/docs/XSTUserGuide.pdf

[3] William N. Joy, Co-Founder, and Vice President for Research on Sun Microsystems, Inc. online: http://www.cs.washington.edu/homes/lazowska/cra/risc.html

[4] Informações genéricas disponíveis online a partir do site: http://www.mips.com/

[5] A single clock cycle MIPS RISC processor design using VHDL Reaz M.B.I, Islam M.S., Sulaiman, M.S., Semiconductor Electronics, Dec. 2002, pp 199 – 203.

[6] Xilinx, online: http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan3_fpgas/

[7] Tutoriais online disponíveis no âmbito da disciplina Sistemas Digitais Reconfiguráveis: http://www.ieeta.pt/~skl/SDR_V.html

[8] P. J. Ashenden, VHDL Tutorial, EDA Consultant, Ashenden Designs Pty. LTD.

[9] Material didáctico da disciplina Arquitectura de Computadores I 2006/2007, acetatos teóricos [17-19]

[10] ISE Xilinx 9.2i homepage online: http://www.xilinx.com/ise/logic_design_prod/foundation.htm

[11] Manual da placa Digilent Inc, Spartan-3 Starter Board online: http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf

[12] Manual de Utilizador da Placa DETIUA-S3, Versão 1.0 por Manuel Almeida, online: www.ieeta.pt/~skl/Research/Projects/Manual_Utilizador_pt.pdf

[13] MIPS Instruction Reference online: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[14] Wikipédia Online: http://en.wikipedia.org/wiki/MIPS_architecture

2. Artigo:

A Tiny Multi-thread RISC MIPS Processor using VHDL

A Tiny Multi-thread RISC MIPS Processor using VHDL

Rui T. Sousa & Bernardo Silva

Resumo – **Este artigo apresenta o estudo e implementação de um processador RISC MIPS Multi-thread usando VHDL. Apenas um subconjunto das instruções nativas do MIPS foi coberto. As arquitecturas RISC são simples em natureza, baseadas numa abordagem load/store, todas as operações são efectuadas sobre operandos residentes nos registos do processador e a memória principal é acedida apenas através das instruções load e store. A arquitectura MIPS tem apenas três tipos de instruções: Tipo-R para computações sobre registos, Tipo-I para computações sobre imediatos e endereços de memória, e Tipo-J para saltos incondicionais. 32 é o número mágico do MIPS, instruções de comprimento fixo de 32 bits, 32 registos de uso geral, words de memória de 32 bits e espaço de endereçamento de memória de 32 bits. Através da adaptação do processador MIPS Single Cycle, foi desenvolvido um processador multi-thread, aproveitando a vantagem de lançar duas instruções de uma só vez, ocupando todos os recursos que estariam parados em operação Single Cycle. Fazendo uso da capacidade das FPGAs modernas é possível desenvolver protótipos de sistemas complexos, detectar problemas ou erros e fazer optimizações, rapidamente e com custos reduzidos, tudo isto graças às ferramentas de software actuais e às linguagens de descrição de hardware, sendo que este projecto tirou grande vantagem dessas capacidades.**

Abstract – **This paper presents a study and implementation of a Multi-thread RISC MIPS processor using VHDL. Only a subset of the native MIPS instructions was covered. RISC architectures are simple in nature, based on a load/store approach, all operations are performed on operands held in the processor registers and the main memory is only accessed through load and store instructions. MIPS architecture has only tree instruction types: R-Type for Register to Register computations, I-Type for Immediate and memory address computations, and J-Type for unconditional Jumps. 32 is the magical MIPS number, 32 bit fixed-length instructions, 32 general-purpose registers, 32 bit memory words and 32 bit memory addressing space. Through adaptation of the Single Cycle MIPS processor, a Multi-thread processor was developed, taking advantage of issuing two instructions at a time, occupying all available resources that were to be idle in a Single Cycle operation. Using the capabilities of modern FPGAs it's possible to develop complex systems prototypes, detect problems or bugs and make optimizations, rapidly and with low costs, all this thanks to current software tools and hardware description languages, this project has greatly taken advantage on those abilities.**

I. INTRODUCTION

Ever increasing demands for speed and throughput on processor computation drive the industry and researchers to find a better architecture for their products and projects.

The Multi-thread approach comes as a follow up to both the Single Cycle and the SuperPipelined MIPS RISC architectures. The latest MIPS architecture designs include an SMT (Simultaneous Multi-Threading) system known as *"MIPS MT"*[1]. A Multi-thread processor is capable of issuing multiple instructions in one machine cycle as opposed to only one in a Single Cycle processor. Performance gains are obvious but the price to pay is increased complexity and resource usage. Modern processors are Superscalar and some are Multi-threaded too, they have many pipeline stages being necessary to deal with data, structural and control hazards, operand forwarding to prevent stalls, dynamic out of order execution and reordering of data and instructions, register content recycling, layers of cache memory and speculative branch assessment[2]. Some of these features fall beyond the scope of this project.

At the time of this article's writing, our Multi-thread MIPS is capable of issuing two instructions simultaneously but from separated processing contexts, an ALU operation and a memory access, a branch and one of the previous or even two branches, a data and control hazards unit was created to govern a pipeline of three stages, a forwarding unit reduces the amount of hazardous situations simplifying the implementation and ease of understanding.

Section II details the general architecture features and our specific datapath's pipeline stages according to their functionality. Section III examines the architecture control components, main control, hazards control and forwarding control. Section IV presents some simulation runs and performance comparison with the Single Cycle MIPS. Section V discusses the results and adds some final remarks. Section VI lists some consulted material for reference.

II. ARCHITECTURE

Our work started out with a Single Cycle MIPS and evolved to incorporate pipelining and Multi-threading techniques, the development platform was the Xilinx Spartan-3 Starter Kit[3], comprising 200.000 equivalent

logic gates and twelve 18Kbit block-RAMs. Implementation was done through VHDL with Xilinx ISE v9.2i software[4].

Using a FPGA as a digital reconfigurable test bed allowed to explore different implementations and easely make corrections and optimizations. A VHDL package was created to allow flexible machine code initialization for the instruction memory, architecture parameters were defined also in the package and the many components code was written accordingly so everything was parameterizable, resource optimization was almost negligible but was an interesting exercise to do.

MIPS architecture has tree instruction types, R-Type for register operations, I-Type for immediate and address computations and J-Type for unconditional branches, all instructions are 32bit in length and have the following bit alignment[5].

R-Type	Field	*opcode*	*rs*	*rt*	*rd*	*sa*	*function*
	Bits	31-26	25-21	20-16	15-11	10-6	5-0
I-Type	Field	*opcode*	*rs*	*rt*	*immediate*		
	Bits	31-26	25-21	20-16	15-0		
J-Type	Field	*opcode*	*instr_index*				
	Bits	31-26	25-0				

Table 1. MIPS instruction formats

Field	Description
opcode	6-bit primary operation code
rd	5-bit specifier for the destination register
rs	5-bit specifier for the source register
rt	5-bit specifier for the target (source/destination) register or used to specify functions within the primary *opcode* REGIMM
immediate	16-bit signed *immediate* used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
instr_index	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary *opcode* SPECIAL

Table 2. MIPS instruction format fields

As stated before only a subset of native instructions was covered as our objectives were to demonstrate the feasibility and do a case study. Most instructions left to implement may be added without significant datapath changes, requiring modifications mainly on the ALU and main control blocks. Table 3 instructions were implemented both on the Single Cycle[6] and the Multi-threaded MIPS for consistency upon comparison.

Most processor architectures share the same basic steps:

1. *Instruction Fetch*; a register (PC-Program Counter) contains a memory address to the current instruction to be executed, the address is fed to the instruction memory and the content is made available to the output port.
2. *Instruction Decode*; some instruction fields enter the control block, depending on these fields a particular instruction is identified and signals flow from the control unit to control what is going to happen.
3. *Operand Fetch*; releases the operands that are stored in the processor's general purpose registers so they can be computed.
4. *Execution*; the ALU receives the operands and performs the computation required by the control unit.
5. *Memory Access*; allows data to be stored in the data memory or loads from it data to be kept on a register.
6. *Write Back*; is the process where data computed by the ALU or read from data memory is stored in the general purpose registers. Finally the PC is updated pointing to the next instruction to be executed.

The previous steps are described in detail according to our implementation for the Multi-threaded MIPS as follows.

Category	Instruction	Name	Format	Assembly code	Description
Arithmetic	add	add	R	add $rd,$rs,$rt	$rd = $rs + $rt
	subtract	sub	R	sub $rd,$rs,$rt	$rd = $rs - $rt
	add immediate	addi	I	addi $rt,$rs,*immediate*	$rt = $rs + *immediate*
Logical	and	and	R	and $rd,$rs,$rt	$rd = $rs & $rt
	or	or	R	or $rd,$rs,$rt	$rd = $rs | $rt
	and immediate	andi	I	andi $rt,$rs,*immediate*	$rt = $rs & *immediate*
	or immediate	ori	I	ori $rt,$rs,*immediate*	$rt = $rs | *immediate*
	shift left logical	sll	R	sll $rd,$rt,*sa*	$rd = $rt << *sa*
	shift right logical	srl	R	srl $rd,$rt,*sa*	$rd = $rt >> *sa*
Data transfer	load word	lw	I	lw $rt,*immediate*($rs)	$rt = Mem[$rs + *immediate*]
	store word	sw	I	sw $rt,*immediate*($rs)	Mem[$rs + *immediate*] = $rt
	load upper immediate	lui	I	lui $rt, *immediate*	$rt = *immediate* × 2^{16}
Branching	branch on equal	beq	I	beq $rs,$rt,*immediate*	if($rs = $rt) goto PC + 4 + *immediate* × 4
Comparison	set on less than	slt	R	slt $rd,$rs,$rt	if($rs < $rt) $rd = 1 else $rd = 0
	set on less than immediate	slti	I	slti $rt,$rs,*immediate*	if($rs < *immediate*) $rt = 1 else $rt = 0

Table 3. MIPS implemented instructions

A. *Instruction Fetch*

A single instruction memory stores the machine code for the two threads, it has a two port address and a two port read interface, refer to figure 1. Supporting two separate threads requires doubling the necessary logic to compute the next PC*X* (for simplicity name*X* is valid for nameA and nameB) for each thread, either a sequential next address by adding 4 bytes or a branch target address to a different location through selection in the multiplexer. The selection signal is a logic *and* of a signal from the control unit that is asserted by the branch *opcode* (Branch*X*) and a signal that is asserted when the contents of `$rs` and `$rt` are the same (BranchTaken*X*) selecting PC*X*_BranchTarget as the NewPC*X*. The PC*X* register is written with the NewPC*X* at the rising edge and the instruction memory is read at the falling edge of the clock signal. Dispatching to the next phase is done at the clock rising edge, through a pipeline register that holds the 32 bit Instruction*X* for decoding and the PC*X* + 4 that is needed to calculate the new target address if a branch condition occurs. At the same time the dispatching is done, the PC*X* register is updated and a new Instruction*X* is soon to be issued.

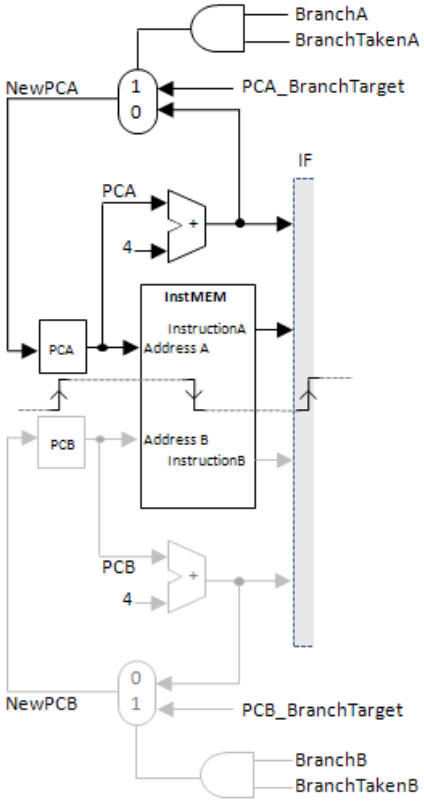


Figure 1. Instruction Fetch architecture

B. *Instruction Decode* and *Operand Fetch*

Instruction decode takes place at the Control Unit, the *opcode* field is the primary identifier for the instruction, the *function* field also takes place in the decoding when

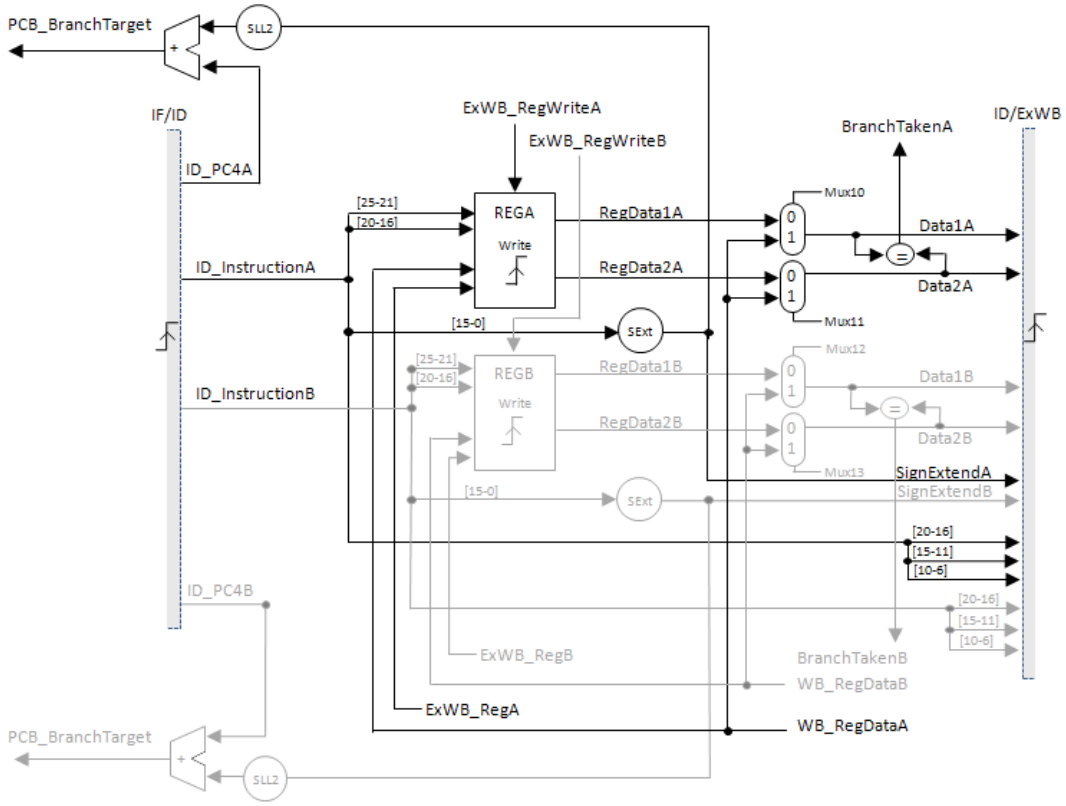


Figure 2. Instruction Decode and Operand Fetch architecture

the instruction is an R-Type one, the *sa* field is necessary to distinguish a `sll` in wich the *opcode* and *function* are 0 from the `nop` (no operation) instruction in wich all 32 bits are 0, if *sa* is different from 0 the instruction is a `sll`. All the necessary control signals used in the *Execution, Memory Access* and *Write Back* phase are stored in the ID/ExWB pipeline registers.

Operand fetch is getting the general-purpose register's contents from both threads out to the datapath, refer to figure 2. Both register files have two 5 bit inputs (*rs* and *rt* instruction fields) that index two of the 32 registers for reading, the contents are outputted in two 32 bit words (RegData1*X* and RegData2*X*), reading is assyncronous and writing is synchronous on the rising edge clock, for writing purposes there is also a 5 bit field to index the register to be written (ExWB_Reg*X*) and a 32 bit port for the data to be stored (WB_RegData*X*).

The lower 16 bits of the instruction, *immediate* field are sign extended to a full 32 bit word used to compute data memory addresses in the *Execution, Memory Access* and *Write Back* phase so it is stored in the ID/ExWB pipeline registers. Signextend*X* is also used to compute the PC*X*_BranchTarget after being multiplied by 4 or equivalently shifted left by two bits so that it is a word offset, recall table 3 description on branching category, this is done to increase the effective range of the offset by a factor of 4[6]. Branch assessment also takes place in the *ID* and *OF* phase, if the branch is taken there is already a fetched instruction that is not going to be executed, so the IF/ID pipeline registers are flushed and one delay slot occurs, if the branch assessment would be located further ahead in the datapath, more pipeline registers would have to be flushed and more delay slots would occur, our implementation only has two pipeline register blocks but the principle stands.

C. *Execution*, *Memory Access* and *Write Back*

The ID/ExWB pipeline registers hold all the necessary control signals and data for execution in the ALU or for computation of the address and data for storing in the data memory. A single bit signal controls all the multiplexers (some are active low) that steer the inputs from thread A or thread B to the designated destination, refer to figure 3.

The ALU execution takes two 32 bit inputs and produces a 32 bit result, operation is selected via a 3 bit control signal, for shift operations there is an additional input indicating the number of bits to be shifted and an additional control bit identifying the shift direction, left or right.

Memory access takes place with the calculation of the data memory address, it's done in a base register plus offset scheme, adding the sign extended *immediate* (SignExtend*X* is renamed on ID/ExWB pipeline register output to ExWB_Offset*X*) to the content of `$rs`. The calculated address points to the memory position to be read or written. The data memory has an input for writing data and an output for reading data to be stored on the

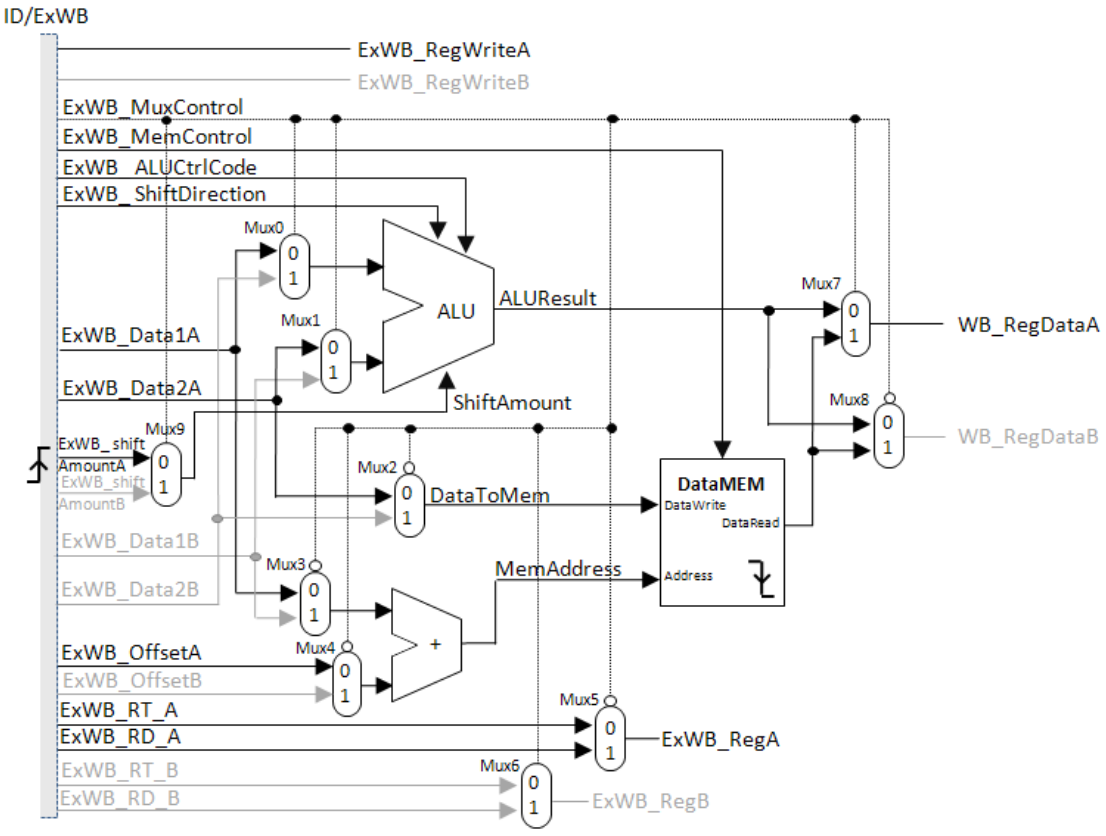


Figure 3. Execution, Memory Access and Write Back architecture

register file, there is an enable control, when the memory is not in use it is disabled, there is also a separate read enable and write enable control for load and store operations, reading and writing are done synchronously at the falling edge of the clock.

The write back data, to be stored on the register file (WB_RegData*X*) is selected from ALU or data memory through a multiplexer, there is one for each thread since both threads have separate register files. Destination register (ExWB_Reg*X*) on the register file is also multiplexed from *rt* or *rd* instruction fields according to the instruction format, *rd* for R-Type and *rt* for I-Type instructions.

III. CONTROL

The control of the entire implementation is done in three units according to their function, main control for the datapath elements, hazard control to prevent wrongfull execution due to structural and control hazards, forwarding control prevents stalling on register dependencies.

D. Main Control

The main control generates signals based on the instructions to execute on both threads, as stated before on section II.*B* the *opcode*, *function* and *sa* instruction fields take place on the decoding hence are connected to the main control, refer to figure 4, there is a signal indicating if the instruction is a branch (Branch*X*) and if `$rs` and `$rt` are equal the branch is to be taken, register file enabling for write operation on an R-Type or load word instruction is asserted by signal RegWrite*X* connected to the ID/ExWB pipeline registers for usage on the next clock cycle, also connected to the pipeline for the same reason are the data memory controls (memory enable, read enable and write enable), the ALU controls that indicate what is to be performed by the ALU, the control for the *Ex Mem* and *WB* multiplexers. The main control also compares the two instructions and signals the hazard control if there is a resource conflict (structural hazard)

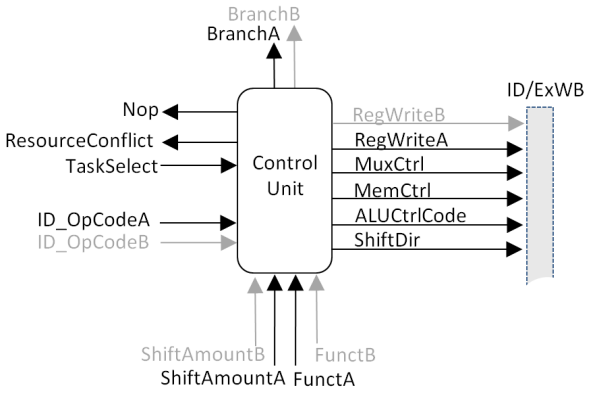


Figure 4. Main Control Unit

when both instructions are R-Type or both access memory, the control unit gets signaled back with TaskSelect to proceed with one instruction and freeze the other. If one or both instructions are `nop` the hazard control also gets signaled.

E. Hazard Control

The hazard control unit manages the PC*X* and pipeline registers IF/ID and ID/ExWB, refer to figure 5.

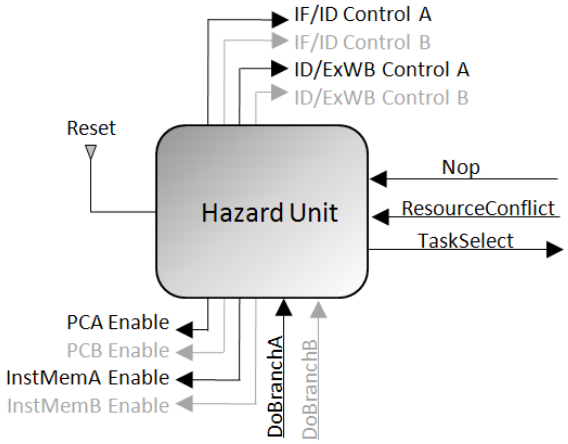


Figure 5. Hazard Control Unit

If there is a resource conflict, one of the program counters and the corresponding instruction memory port are disabled and the instruction is freezed, there is no priority scheme so alternating execution on structural hazards is the fair way to decide wich thread goes on (TaskSelect).

If a `nop` is detected, the PC*X* stays enabled but the IF/ID pipeline is disabled and the ID/ExWB pipeline is flushed for that thread, there is no problem when flushing pipeline registers because it occurs synchronously and any signal necessary to the next stage is already stable and used correctly.

In the case of a taken branch, both pipeline registers are flushed, and again, any data to be stored on the register file gets where it should.

F. Forwarding Control

The forwarding control resolves the data hazards caused by register dependencies, figure 6 ilustrates an example.

In a pipelined flow some instructions may require data that is not yet available in the register files, it would be necessary to wait for the data to be stored in the register file and fetched afterwards, this wait is generally called bubbles, like a bubble preventing the flow in a pipe.

Forwarding takes place when the register that is going to be written is the same as one to be read in the subsequent instruction, then the data is placed in the pipeline registers immediately before the execution phase, becoming

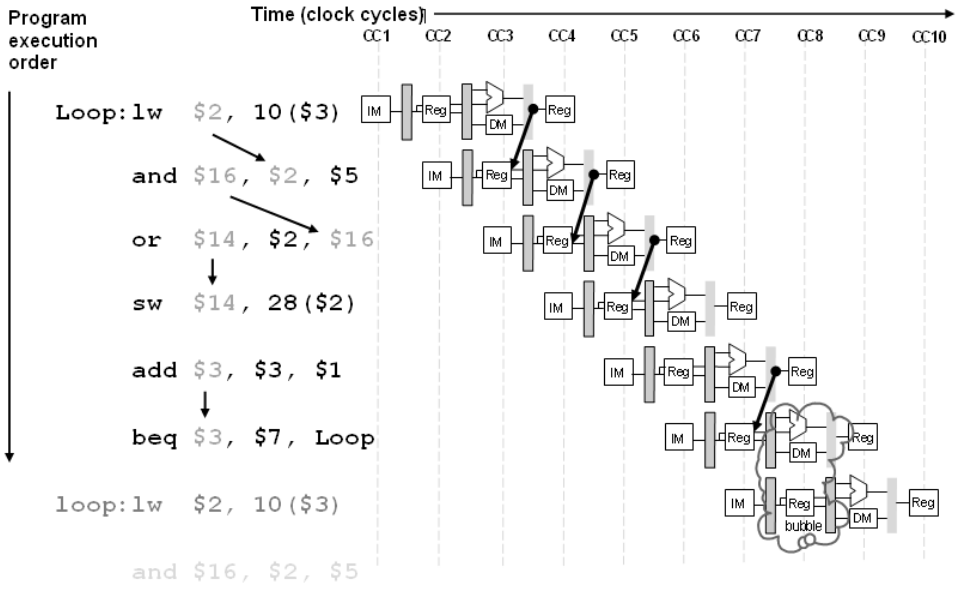


Figure 6. Example of register dependency resolved by forwarding

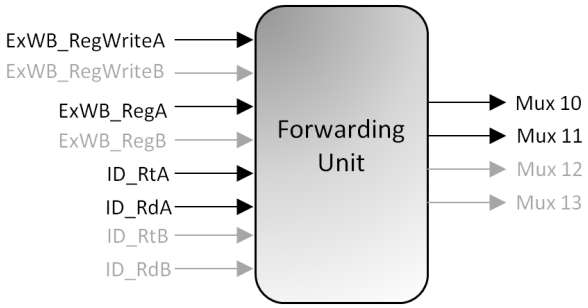


Figure 7. Forwarding Control Unit

available earlier than writing in the register file and operand fetch, refer to figure 2 and notice multiplexers Mux10 through Mux13.

PCA (hex)	Thread A instructions	PCB (hex)	Thread B instructions
00	lw $1,0($0)	44	nop
04	lw $3,4($0)	48	nop
08	lw $4,8($0)	4c	nop
0c	lw $5,20($0)	50	nop
10	lw $7,10($0)	54	nop
14	loop:lw $2,0($3)	58	add $1,$0,$0
18	add $4,$4,$2	5c	lw $2,0($0)
1c	sw $4,24($0)	60	lw $4,24($0)
20	add $3,$3,$1	64	add $4,$2,$4
24	slt $6,$3,$5	68	sw $4,28($0)
28	beq $6,$7,loop	6c	lw $8,28($0)
2c	sw $4,28($0)	70	nop
30	lw $8,28($0)	74	stop:beq $0,$0, stop
34	stop:beq $0,$0,stop	78	add $4,$4,$4
38	nop	7c	nop

Table 4. Instruction Memory

IV. SIMULATION AND ANALISIS

The simulation engine was the standard built in simulator from the Xilinx ISE v9.2i. To test the Multi-thread implementation the instructions in table 4 were used.

Thread A performs the sum of an array of four elements from the data memory, stores the result in memory at a specified address and loads it back in the register file. Thread B loads two values from data memory, one of them is previously written by thread A, sums them and also stores the result in memory and back in the register file.

The code was specifically written to cause some hazards as presented in figure 8.

Address (hex)	Memory content (hex)
00	4
04	10
08	0
0c	8
10	1
14	2
18	3
1c	4
20	20
24	55
28	0
2c	0

Table 5. Data Memory

Table 5 shows the data memory contents at simulation start.

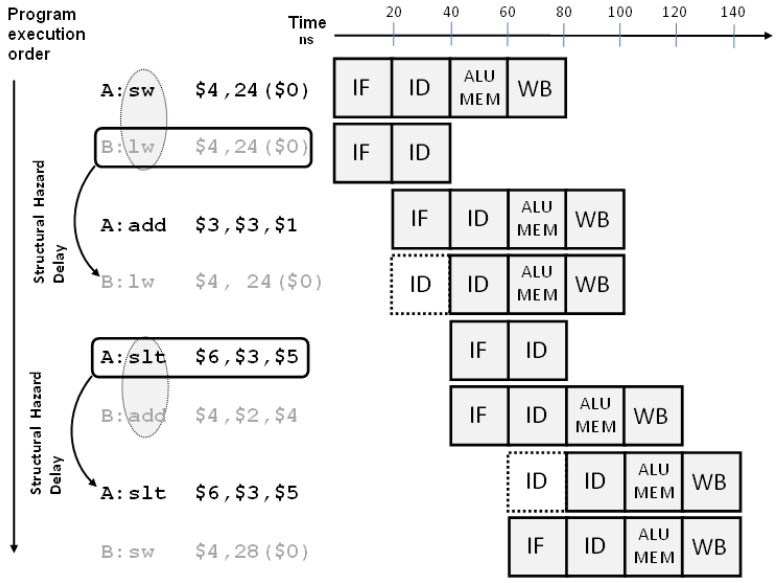


Figure 8. Structural hazard resolving

From the resulting simulation, some signals of interest are shown in figure 9, instructions addressed by PCA=1C a `sw` and PCB=60 a `lw` result in a structural hazard, both try to access the data memory, notice highlight A, ResourceConflict goes high, TaskSelect is low meaning thread A will proceed and thread B gets freezed so PCB stays the same for another clock cycle. Another structural hazard occurs at instructions addressed by PCA=24 a `slt` and PCB=64 a `add`, this time both try to use the ALU, now it is thread B's turn to proceed and thread A gets freezed shown by highlight B in figure 9. Highlight D shows the branches taking place and highlights E1 and E2 show the final results of thread A (`$8`=A) and thread B computing (`$8`=5). Since clock cycle 15 thread B is on a branch to itself, the instruction at PCB=78 is fetched and decoded as DoBranchB is not always high but the branch is always taken so the pipeline register gets flushed and that `add` instruction is never executed, this can be seen also in highlight E2 as `$4` never changes onward.

V. CONCLUSIONS AND FINAL REMARKS

The architecture described was implemented on the Spartan-3 Starter Kit after successful correct simulation, for validation the boards four 7 segment displays were used as a visual aid showing both program counters contents, it was possible to view when one of the threads was frozen due to structural hazard and the other thread proceeded, the boards 8 leds were also used to display the contents to be written in the register files for both threads, enabling confirmation of the computed results.

Device utilization is summarized in table 6 for comparison between the Single Cycle implementation and the Multi-thread one.

No performance measurement was done except for the test code of table 4, in the Single Cycle processor, if both programs from the two threads were executed sequentially it would take 37 clock cycles to completion excluding the `nops` and the stop label branches for both threads, figure 9 shows completion takes also 37 clock cycles but thread B completes in 15 cycles even counting with the first five `nops` so resource utilization is not optimal for the rest of the remaining 22 cycles it takes for thread A to finish. Longer test programs would yield better results favoring the Multi-thread processor, even in the presence of some unavoidable structural hazards.

Future work will undoubtly pass through integrating J-Type instructions, `j` jump, `jr` jump register and `jal` jump and link, other R-Type and I-Type instructions, addition of a multiplyer, devider, floating poin unit and support for system calls.

Upon implementation of the mentioned lacking features, this processor is a good candidate for FPGA based embedded systems that relly heavily on parallel processing, for example, one thread might be used for unstopped main program computation while the other thread deals with interrupt service routines asked for by some peripherals.

Selected Device : 3s200ft256-5								
	Single Cycle				Multi-thread			
Number of Slices:	472	out of	1920	24%	882	out of	1920	45%
Number of Slice Flip Flops:	8	out of	3840	0%	306	out of	3840	7%
Number of 4 input LUTs:	998	out of	3840	25%	1889	out of	3840	49%
Number used as logic:	742				1377			
Number used as RAMs:	256				512			
Number of IOs:	3				3			
Number of bonded IOBs:	3	out of	173	1%	3	out of	173	1%
Number of BRAMs:	2	out of	12	16%	2	out of	12	16%
Number of GCLKs:	2	out of	8	25%	2	out of	8	25%
Number of DCMs:	1	out of	4	25%				

Table 6. Device utilization summary

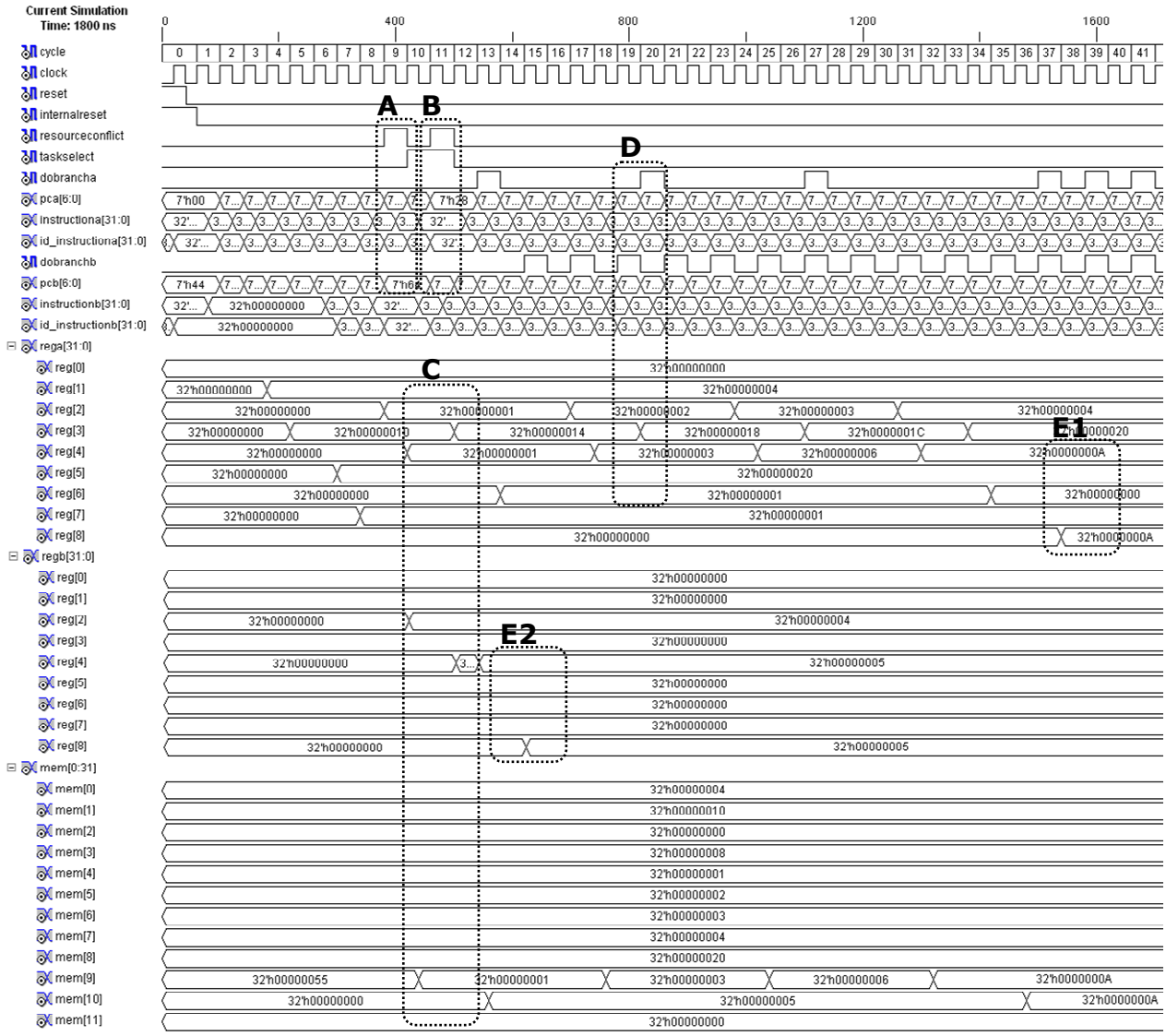


Figure 9. Simulation run of the implemented design

VI. REFERENCES

[1] MIPS web site, MIPS MT architecture
http://www.mips.com/products/architectures/mips-mt-ase/

[2] John P. Shen, Mikko H. Lipasti,
"Modern Processor Design" - McGraw Hill

[3] Spartan-3 Starter Kit Board User Guide
http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf

[4] Xilinx ISE Webpack
http://www.xilinx.com/ise/logic_design_prod/webpack.htm

[5] MIPS32® Architecture for Programmers Volume I - Introduction to the MIPS32® Architecture, MIPS32® Architecture for Programmers Volume II - The MIPS32® Instruction Set
http://www.mips.com/products/resource-library/product-materials/mips-architecture/

[6] David A. Patterson, John L. Hennessy,
"Computer Organization & Design – The Hardware/Software Interface" – Morgan Kaufmann Publishers, Inc.

Other consulted resources not referenced

- The *XST User Guide:*
http://wsl.cemed.ua.pt/moodle/file.php/4/docs/XSTUserGuide.pdf

- Wilipedia, the free encyclopedia
http://en.wikipedia.org/wiki/MIPS_architecture
http://en.wikipedia.org/wiki/Simultaneous_multithreading

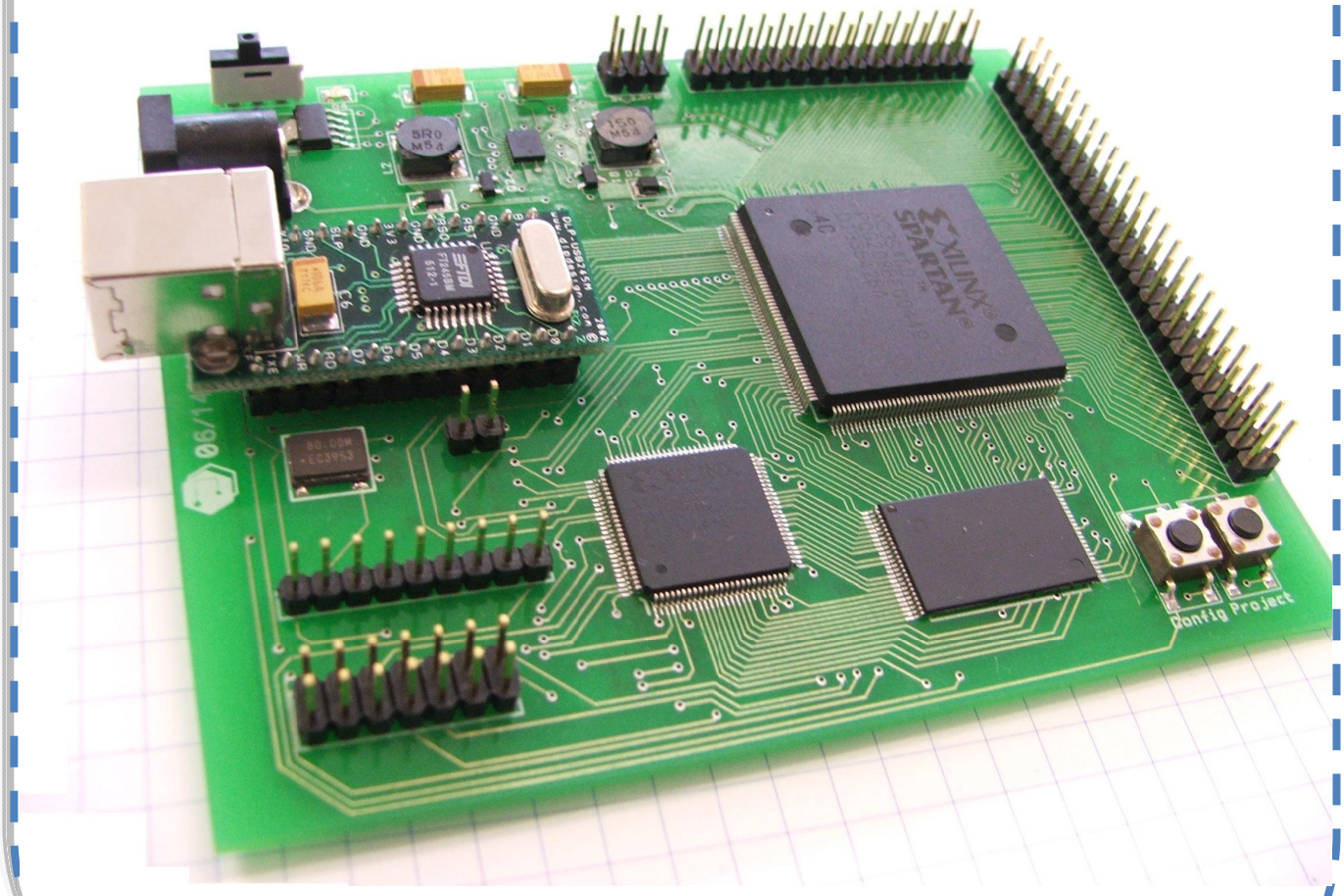
- Simultaneous multi-threading resources, many articles and presentations at
http://tbp.berkeley.edu/~jdonald/research/hyperthreading/

3. Tutorial de configuração da placa DETIUA-S3

Universidade de Aveiro

Tutorial de Configuração
da
Placa DETIUA-S3

Versão 1.1



Bernardo Silva
Bernardo.Silva@ua.pt
Março de 2008

Prefácio

Este documento tem como objectivo a descrição sequencial de quais os passos, que medidas e cuidados a tomar para (re)programar a Placa DETIUA-S3.

A (re)programação consiste em reconfigurar o CPLD e/ou FPGA com as BitStreams iniciais, ficando assim a placa pronta a efectuar a comunicação com o interface USB e receber novas BitStreams desenvolvidas.

Se não conseguir encontrar solução ao seu problema neste documento, procure na bibliografia outras referências nas quais poderá consultar e eventualmente possuir resposta à informação que procura.

AVISO: *Antes da configuração do CPLD e/ou FPGA com o Default BitStream, deve ser feita uma exposição do motivo da necessidade de reprogramação e verificação do material a um docente ou responsável pelo material.*

Índice

Material Necessário

Para efectuar o procedimento de configuração do CPLD e/ou FPGA da placa DETIUA-S3, é necessário o material que se encontra na seguinte tabela (Tabela 1).

Descrição do Material	Ilustração
Computador com: • **ferramenta iMPACT da Xilinx (instalada conjuntamente com o software "Xilinx ISE" [até versão 7.1])** • **capaz de suportar 2 ligações USB**	
Xilinx Multilinx USB Cable e respectivos drivers [4] correspondentes ao sistema operativo.	
Aplicação "Prototyping Board Manager" (versão >= 1.0)	
Dois cabos USB A-B (Macho-Macho)	
Um fio auxiliar em cobre (apenas descarnado nas pontas)	
Placa DETIUA-S3 e respectivos drivers [5] da sua interface USB	
Ficheiros de configuração do CPLD (CPLD.jed) e Default BitStream (DEFAULT_BITSTREAM.bit)	

Tabela 1 - Breve descrição do material necessário.

NOTA: Todo este material pode ser encontrado no laboratório 0.24 do Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA) e possível de ser usado mediante a autorização dos respectivos responsáveis pelo conteúdo do laboratório. Software, drivers e ficheiros de configuração necessários encontram-se instalados num dos computadores do laboratório utilizando a conta de utilizador "CSLAb" (deverá ser pedida a *Password* para utilizar esta conta).

Breve Descrição do Hardware e Ferramentas Usadas

DETIUA-S3

Localização dos vários componentes integrados na placa, referidos neste manual necessários para efectuar a configuração:

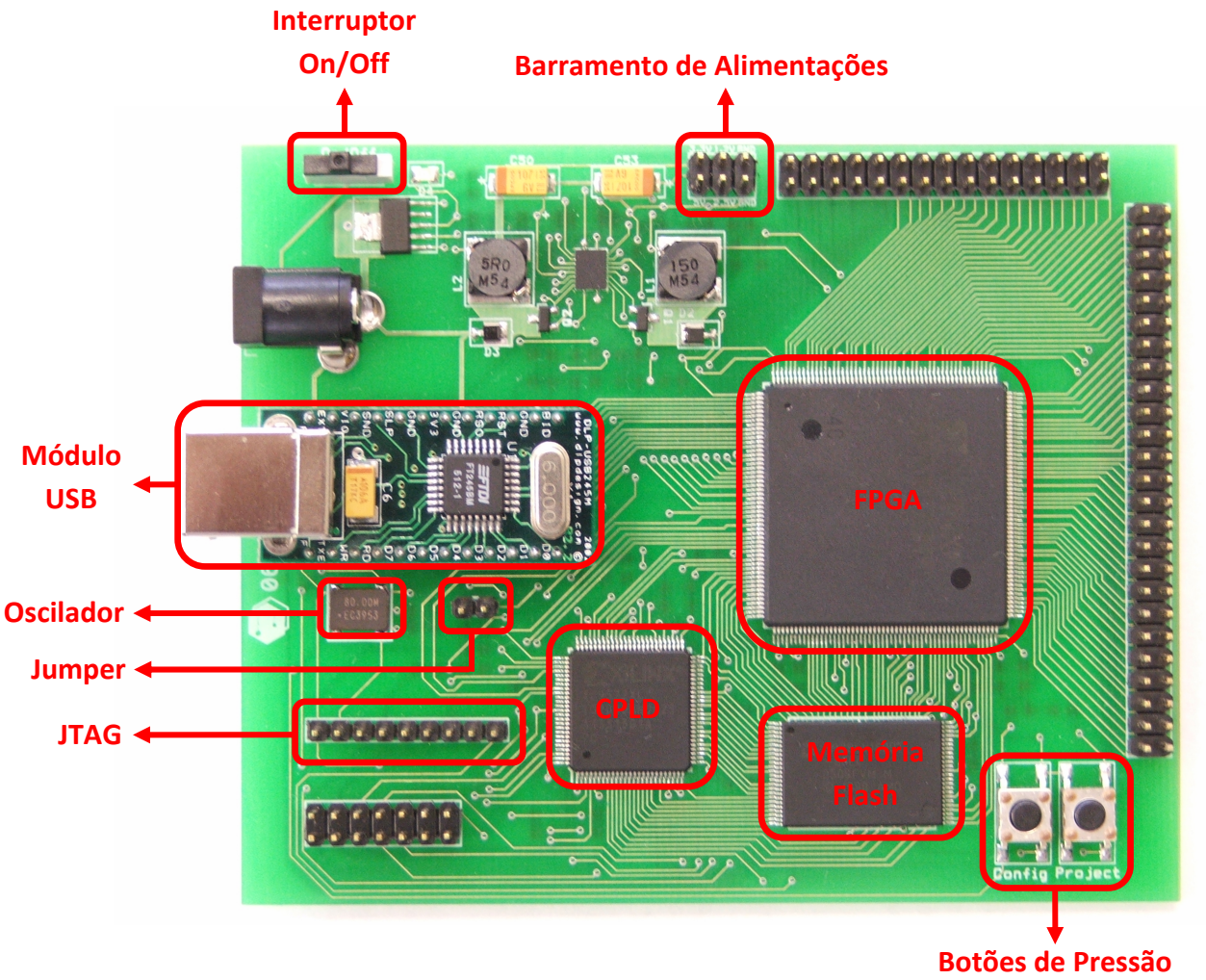


Figura 1 – Localização dos componentes na placa

- **Interruptor On/Off**
  Interruptor que Liga ou Desliga a placa. Quando ligada, possui um led vermelho situado à sua direita que acende.

- **Barramento de Alimentações**
  Barramento capaz de fornecer 1.2V, 2.5V, 3.3V e 5V de alimentação a placas de extensão possíveis de acoplar à placa DETIUA-S3.

- **Módulo USB**

O módulo USB utilizado foi o DLP-USB245M, que é um interface USB-FIFO paralelo. Este módulo da empresa DLP Design, tem como componente central o FT245BM da FTDI.

- **Jumper**

Para que se possa reprogramar a primeira zona de memória flash correspondente à Default BitStream, será necessário adicionar um *Jumper* para que seja permitida a escrita nessa zona.

- **Oscilador**

A placa utiliza um oscilador de 80 MHz. Este sinal de relógio é utilizado pela FPGA e pelo CPLD, num dos pinos de global clock.

- **JTAG**

A placa está dotada de um conector que disponibiliza pinos dedicados, que são usados para configurar a FPGA e o CPLD. Este modo de configuração Boundary Scan, mais conhecido como JTAG, permite uma configuração directa sendo apenas necessário o cabo multilinx e a ferramenta IMPACT da Xilinx.

Ao utilizar o JTAG, vai aparecer uma corrente de configuração onde aparece primeiro o CPLD e posteriormente a FPGA. Nesta forma, o primeiro dispositivo a ser possível configurar é o CPLD. Para configurar a FPGA não é necessário ter que configurar o CPLD, devido existir uma opção na ferramenta IMPACT que possibilita saltar para o próximo dispositivo.

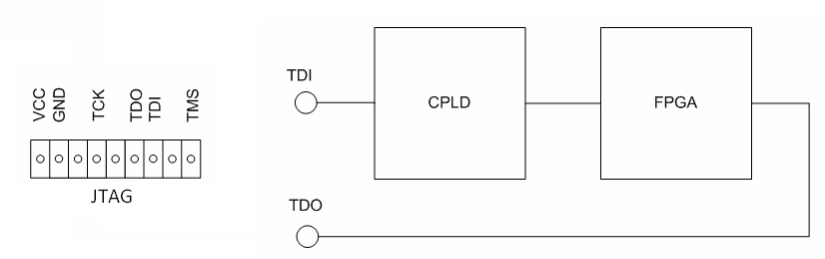


Figura 2 – À esquerda a representação esquemática dos pinos da JTAG; à direita representação esquemática da interligação do CPLD e FPGA

- **Botões de Pressão**

Existem dois botões de pressão na placa, um denominado de *Config* e o outro *Project*.

O botão *Config* permite ao utilizador configurar a placa com um ficheiro de configuração por defeito (usualmente o ficheiro .bit que contém o projecto desenvolvido).

O botão *Project* permite ao utilizador disparar a execução do ficheiro transferido.

- **CLPD**

A placa contém um CPLD XC9572XL da Xilinx, que é utilizado no processo de configuração da FPGA.

A memória flash contém o ficheiro de configuração (bitstream) da FPGA, sendo necessário um componente, neste caso o CPLD, que realize o controlo da memória flash de modo a permitir a configuração da FPGA [1].

- **FPGA**

A FPGA Spartan-3 da Xilinx (XC3S400-4-PQ208) utilizada como componente reconfigurável, contém 400 mil portas de sistema, 56Kb de RAM distribuída, 16 multiplicadores, 141 pinos de entrada e saída, etc.

- **Memória Flash**

A placa utiliza uma memória *flash* de 16 Mbit com a referência AM29LV160. Esta memória é constituída por 31 sectores de 64Kbytes, 1 de 32Kbytes, 1 de 16Kbytes e 2 de 8Kbytes.

A memória está divida em 3 zonas como ilustra a imagem à direita, sendo a primeira onde se localiza a Default BitStream, na segunda a BitStream do Utilizador e uma terceira disponível para guardar dados ou outras BitStreams (figura 3).

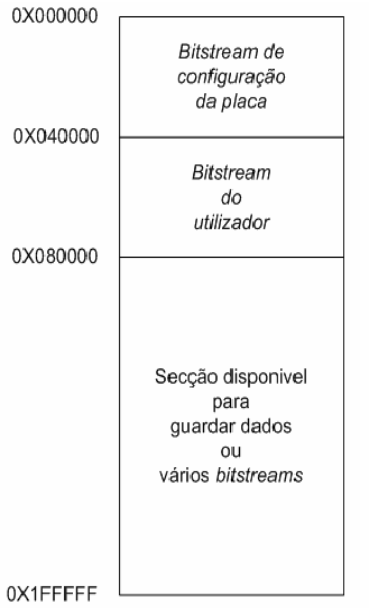


Figura 3 – Divisão e endereços iniciais das zonas constituintes da memória.

AVISO: *Se por algum motivo a primeiro sector da memória flash for alterado a placa não funcionará correctamente, ou ficará inutilizada, sendo a aplicação deste tutorial uma possível solução do problema.*

Cabo Multilinx USB da Xilinx

Pode-se usar o cabo MultiLINX para efectuar o download e verificar todos os CPLDs e FPGAs da Xilinx. O hardware do cabo MultiLINX comunica com o anfitrião através do Universal Serial Bus (USB) a uma taxa de transferência de 12Mbit/s ou numa taxa variável através de uma ligação de interface RS-232 a uma velocidade de 57600bits/s.

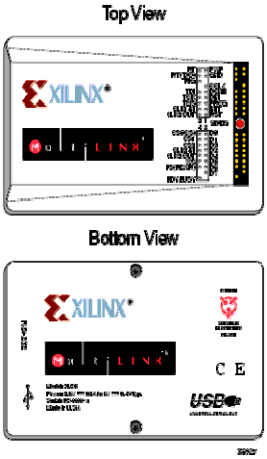


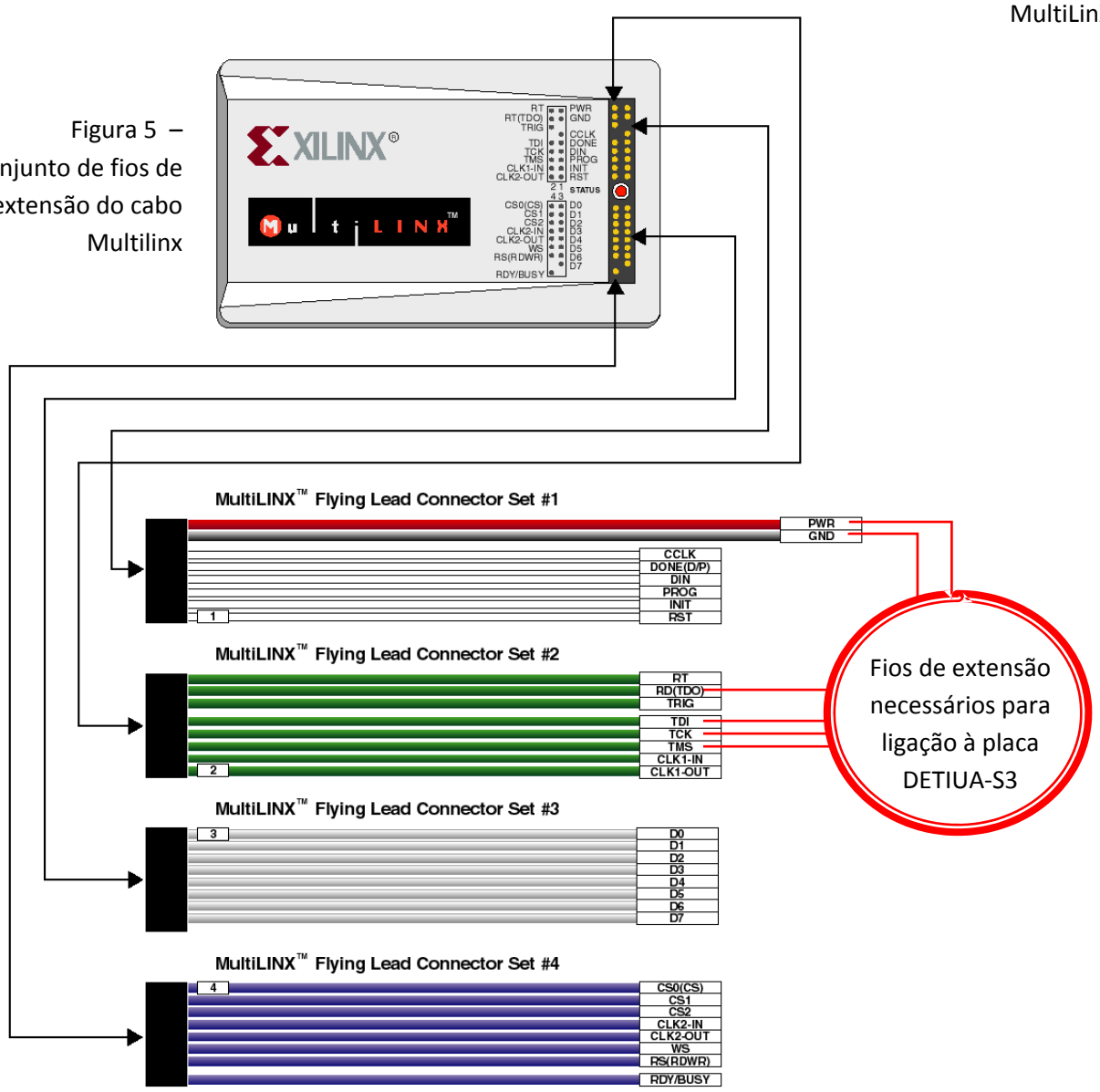
Figura 4 – Vista superior e inferior do dispositivo MultiLinX.

Requisitos de alimentação do cabo MultiLINX :

Este cabo é alimentado através da placa que contém o circuito do utilizador, neste caso, a DETIUA-S3. A ligação USB não oferece qualquer tipo de alimentação, sendo assim necessário ligar o fio vermelho (PWR) e preto (GND) ao circuito. O mínimo de voltagem de entrada para o cabo é de 2.5V (com um consumo de .8A) e o máximo é de 5V (.4A).

Fios de extensão do cabo Multilinix:

Figura 5 – Conjunto de fios de extensão do cabo Multilinx



Xilinx iMPACT

iMPACT permite que projectistas facilmente programem ou configurem dispositivos ou por uma consola, introduzindo instruções ou de uma forma mais conveniente através de uma interface gráfica.

É uma ferramenta usada para configuraçao e programação de todos os Xilinx PLDs (FPGAs e CPLDs) e PROMs. Este ferramenta pode-se encontrar incluida no pacote de software de design de hardware ISE™. [2]

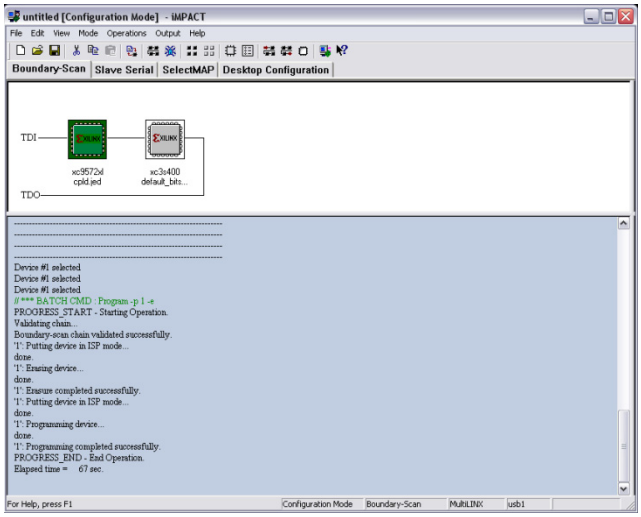


Figura 6 – Ambiente gráfico da ferramenta Xilinx iMPACT

Prototyping Board Manager (PBM)

Ferramenta complementar no uso do software de gestão da placa de prototipagem DETIUA-S3. Através desta ferramenta é possível efectuar operações de manutenção, transferência de dados e obtenção de dados da placa DETIUA-S3. [3]

Figura 7 – Última versão disponível do software PBM

Preparação e Interligação do Material

Para que seja possível a configuração o material deve ser montado e manuseado com extrema atenção.

AVISO: *Qualquer tipo de erro de ligação incorrecta ou indevida não declarada neste manual, pode conduzir à avaria ou mesmo à inutilização da Placa DETIUA-S3.*

Deve-se seguir os seguintes passos, na ordem apresentada, de modo a garantir o sucesso de (re)configuração da placa:

Passo 1

- Adicione o Jumper (figura 8 – Passo 1).

Passo 2

- Ligar:

a) o fio "PWR" de extensão do cabo Multilinx com o barramento de alimentação (Figura 8 – Passo 2 e Esquema 1)

b) Fio auxiliar, necessário posteriormente para configuração da placa com a Default BitStream (***fica ligado apenas em uma das extremidades, sem que a outra fique em contacto com mais nada***) (Figura 8 – Passo 2 e Esquema 1).

Passo 3

- Ligar os fios de extensão aos pinos da JTAG (Figura 8 – Passo 3 e Esquema 2).

Passo 4

- Ligar os dois cabos USB às entradas USB do computador e ligar a placa DETIUA-S3 (LED vermelho da placa e verde do cabo acendem).

Passo 5 (Opcional)

- Se no computador que estiver a efectuar a configuração, é a primeira vez que está a ligar a placa DETIUA-S3 ou o cabo MultiLINX, deverá aparecer a notificação de novos dispositivos USB encontrados. Deverá proceder à instalação dos respectivos drivers para cada um dos componentes antes de prosseguir [4 e 5].

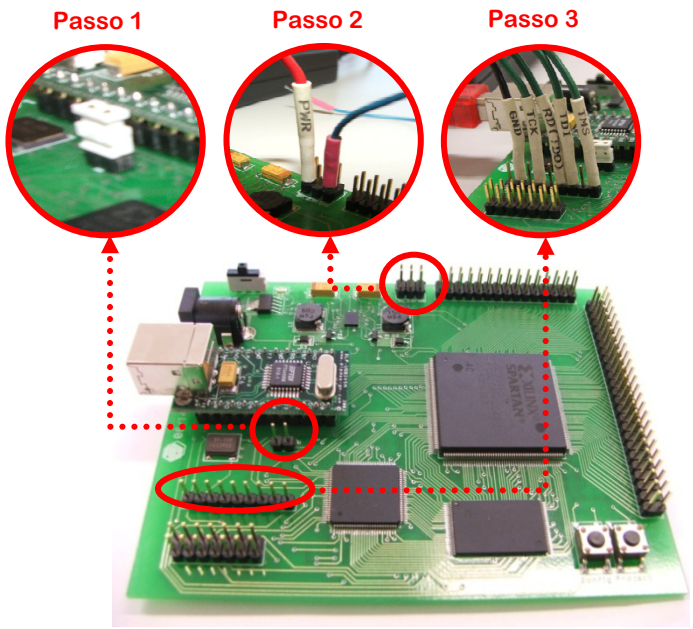
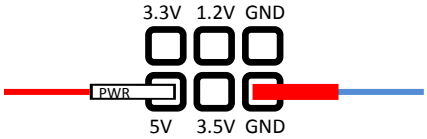
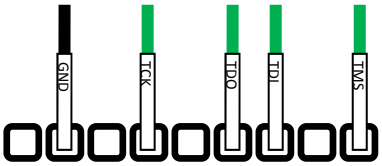


Figura 8 – Ligações efectuadas na placa



Esquema 1 - Ligações no Barramento de Alimentação



Esquema 2 - Ligações no barramento JTAG

AVISO: *Deve-se verificar novamente no final de todos os passos se tudo está devidamente ligado no local correcto e bem configurado.*

Configuração do CLPD e/ou FPGA

Após a montagem e verificação das ligações dos vários cabos com a placa, pode seguir os seguintes passos para transferir os ficheiros de configuração para o(s) componente(s) pretendidos:

Passo 1
– Ferramenta iMPACT

- Abrir a ferramenta "iMPACT" da xilinx [Figura 9]

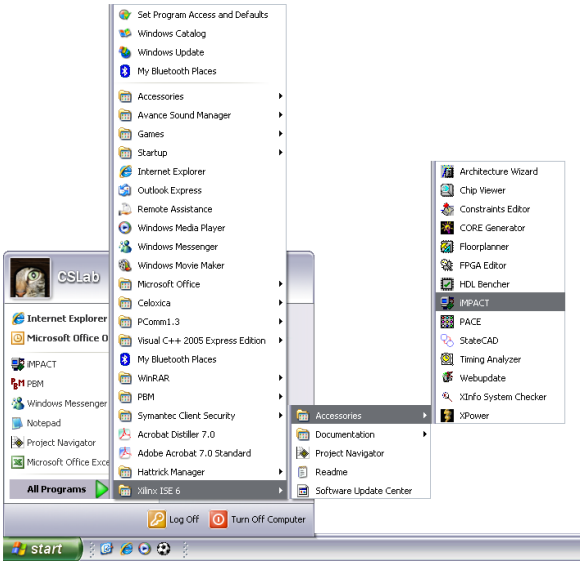


Figura 9 - Menu *Start → Xilinx ISE 6 → Accessories → iMPACT*

- No menu que aparece escolher "Configure Devices" e clicar em "*Next*" [Figura 10];

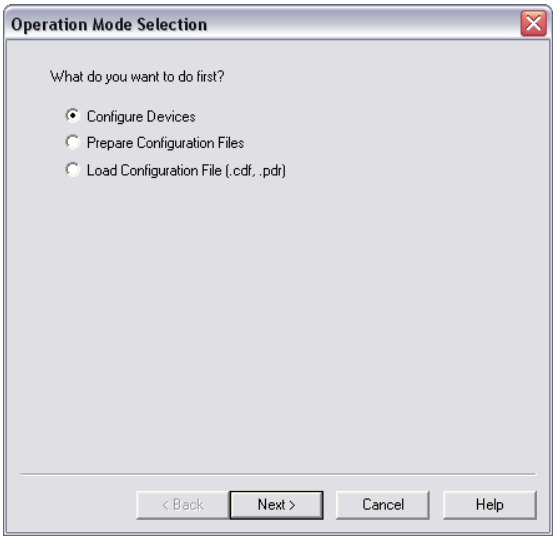


Figura 10 - Operation Mode Selection

- Na janela "Configure Devices" seleccionar "Boundary-Scan Mode" e clicar em "Next" [Figura 11];

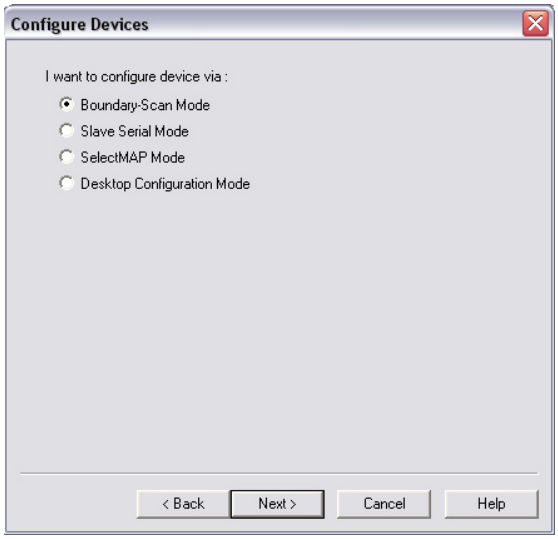


Figura 11 – Configure Devices

- Na janela "Boundary-Scan Mode Selection", escolher a opção "Automatically connect to cable and identify Boundary-Scan chain" e clicar em "Finish" [Figura 12];

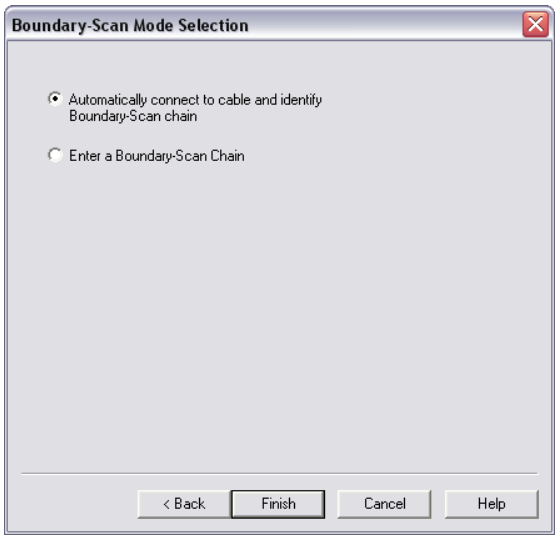


Figura 12 – Boundary-Scan Mode Selection

- Se tudo correu bem na comunicação com a placa DETIUA-S3, deve-se verificar as seguintes condições:
    - Deve aparecer um menu OK: "Boundary-Chain Scan Contents Sumary" informando que foram dectectados "… 2 devices …";
    - Na janela principal do iMPACT deve-se visualizar quais são os dois componentes detectados (esquema de ligação semelhante ao da figura 13)
    - Os dois componentes detectados têm de possuir os seguintes nomes:
        - "xc9572xl" – Correspondente ao CPLD
        - "xc3s400" – Correspondente à FPGA

***NOTA:* Se não forem verificadas estas condições, deverá fechar por completo todas as janelas abertas do iMPACT, e recomeçar o Passo 1. Isto pode acontecer mais frequentemente, pois basta um mau contacto com uma das ligações ou alguma perda de informação durante o processo de comunicação. NÃO AVANCE, enquanto todas as condições não se verificarem.**

Passo 2
– Selecção das BitStreams

Estando o passo 1 completado, prossegue-se a configuração.

- Clique no botão “OK” da caixa “Boundary-Chain Scan Contents Summary”.

É neste ponto que pode optar por **Configurar** ou **Não Configurar** o CPLD. Verifique que se encontra a verde o bloco mais à esquerda, correspondente ao CPLD, no esquema de interligação CPLD ↔ FPGA.

- **Para configurar o CPLD** deve utilizar a janela “Assign New Configuration File” para escolher o ficheiro: “CPLD.jed” (no caso do computador do Lab 0.24 do IEETA, este ficheiro encontra-se no “Desktop”) e clicar em “Open” [Figura 13];
- **Se NÃO pretende configurar o CPLD** deve apenas clicar no botão “Bypass” [Figura 13];

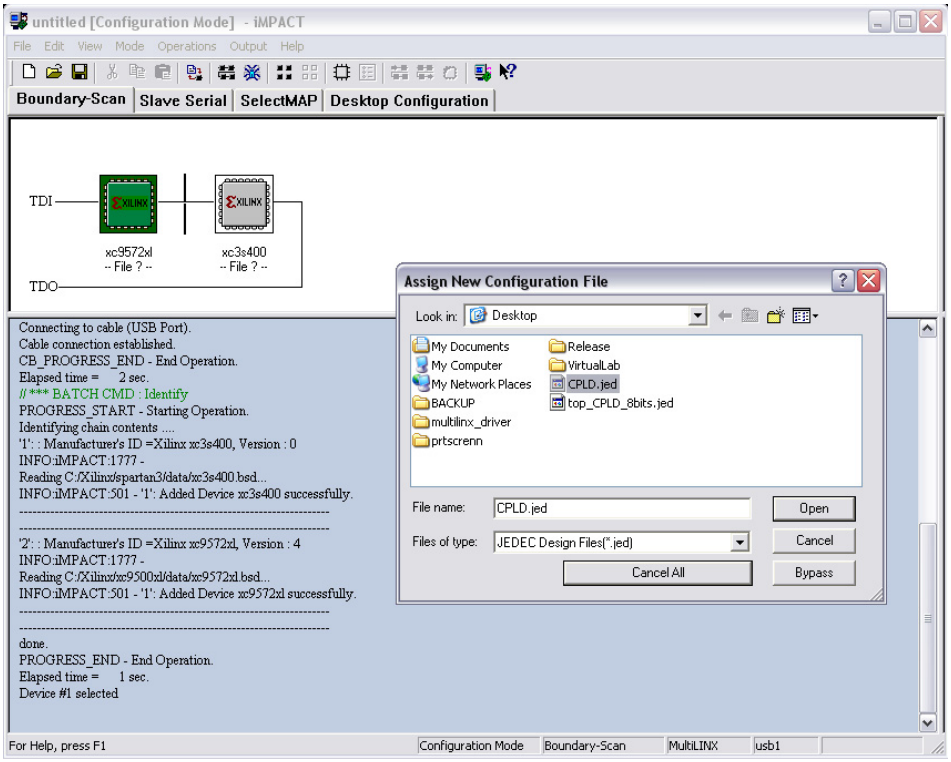


Figura 13 – “Assign New Configuration File” para CPLD

- Surge uma nova janela em toda idêntica à anterior mas agora para seleccionar o ficheiro de configuração da FPGA. Verifique na figura 14, comparativamente à figura 13, que agora encontra-se a verde o bloco correspondente à FPGA.
  Do mesmo modo, que anteriormente, pode ou não configurar a FPGA com a Default BitStream.
  - Se pretender programar a FPGA com a Default BitStream então deve procurar seleccionar o ficheiro “DEFAULT_BITSTREAM.bit” do disco rígido da máquina local (novamente, se estiver a utilizar o computador do Lab 0.24 do IEETA este ficheiro encontra-se no “Desktop” com o nome anteriormente mencionado) e fazer clique em “Open” [Figura 14];
  - Se NÃO PRETENDER configurar a FPGA com a Default Bit Stream, clique em “Bypass” [Figura 14];

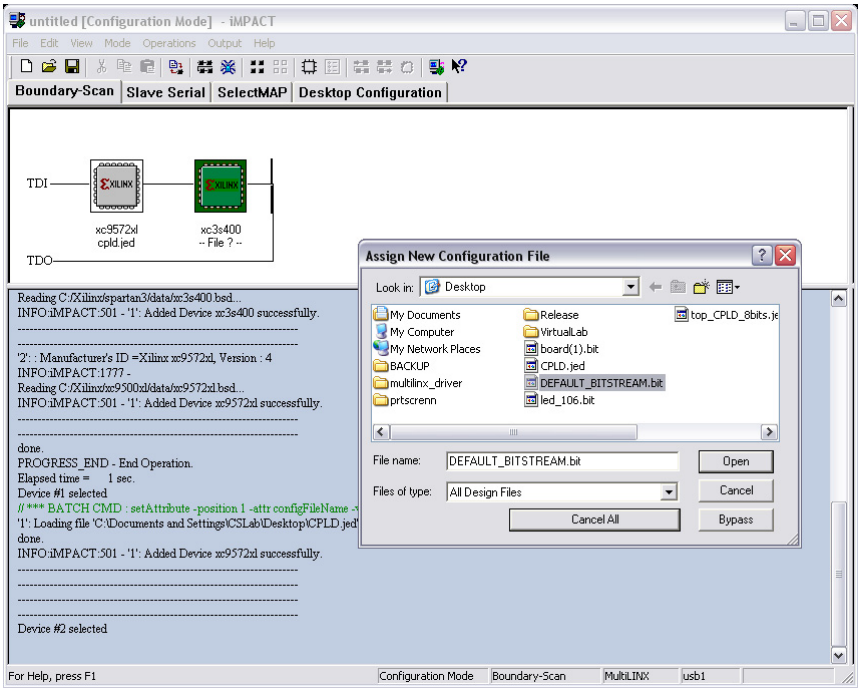


Figura 14 – "Assign New Configuration File" para FPGA

- É possível que surja uma mensagem de alerta relativamente ao "Startup Clock", ignore, e clique em "OK" [Figura 15];

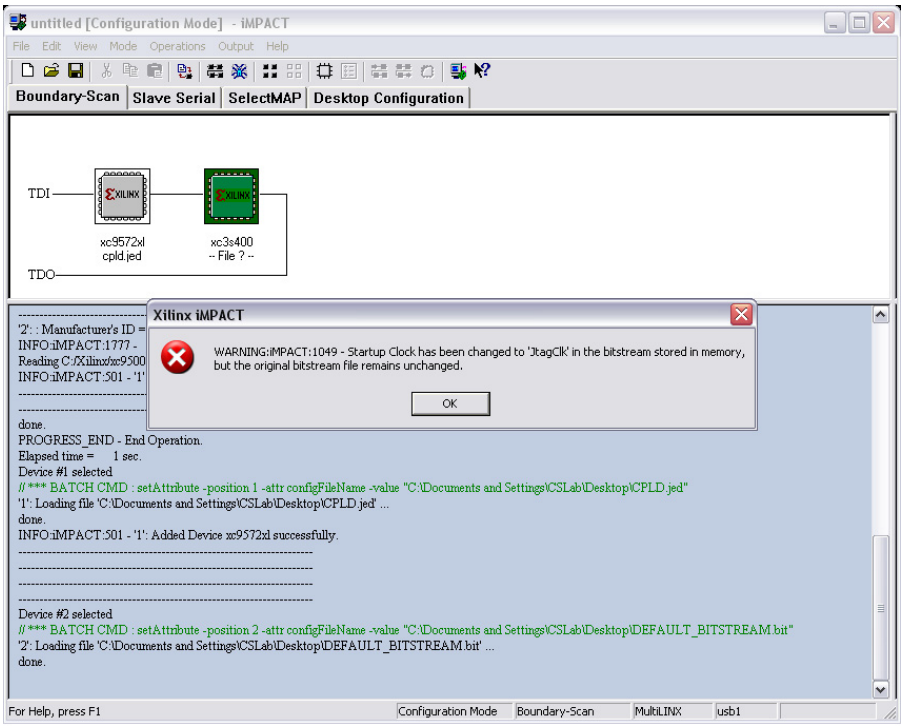


Figura 15 - Xilinx iMPACT aviso sobre mudança do "Startup Clock"

Neste momento conclui-se o passo 2 e encontra-se tudo pronto para iniciar a transferência de dados para a placa.

Passo 3

- Programação do CPLD

Se escolheu um ficheiro para configurar o CPLD então, clique com o botão direito sobre o bloco correspondente ao CPLD, e surgirá um menu com os comandos possíveis, escolha “program…” (Figura 16)

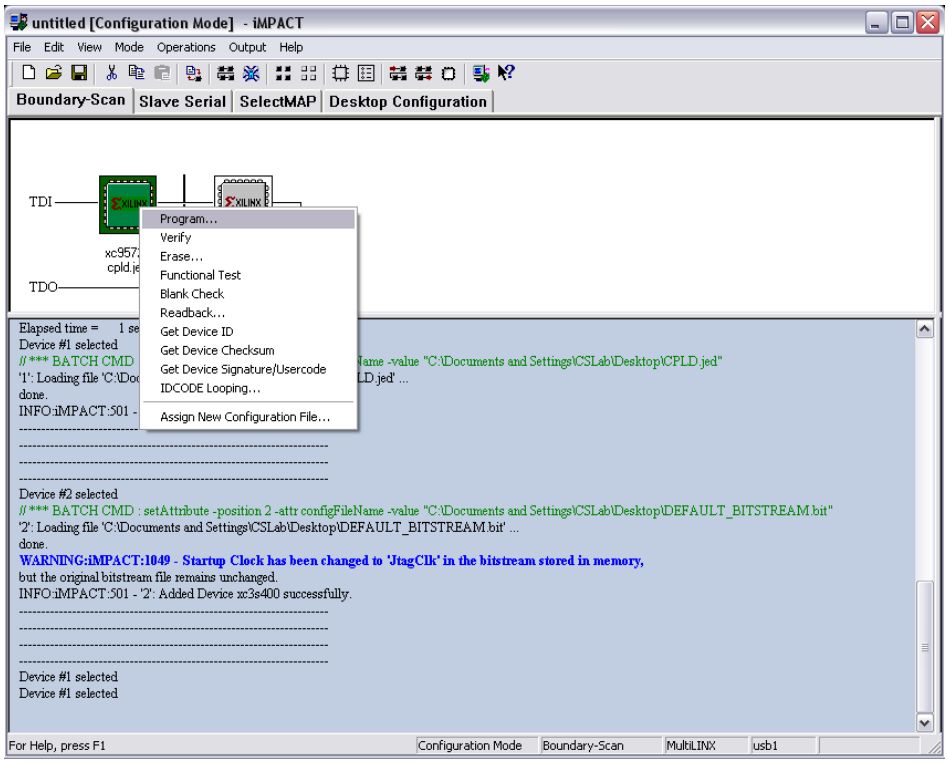


Figura 16 – Menu do botão direito do rato sobre o CPLD na janela do iMPACT Xilinx

No menu que surge, ponha um visto apenas na opção “Erase before programing” para apagar a programação existente e configurar com a nova.

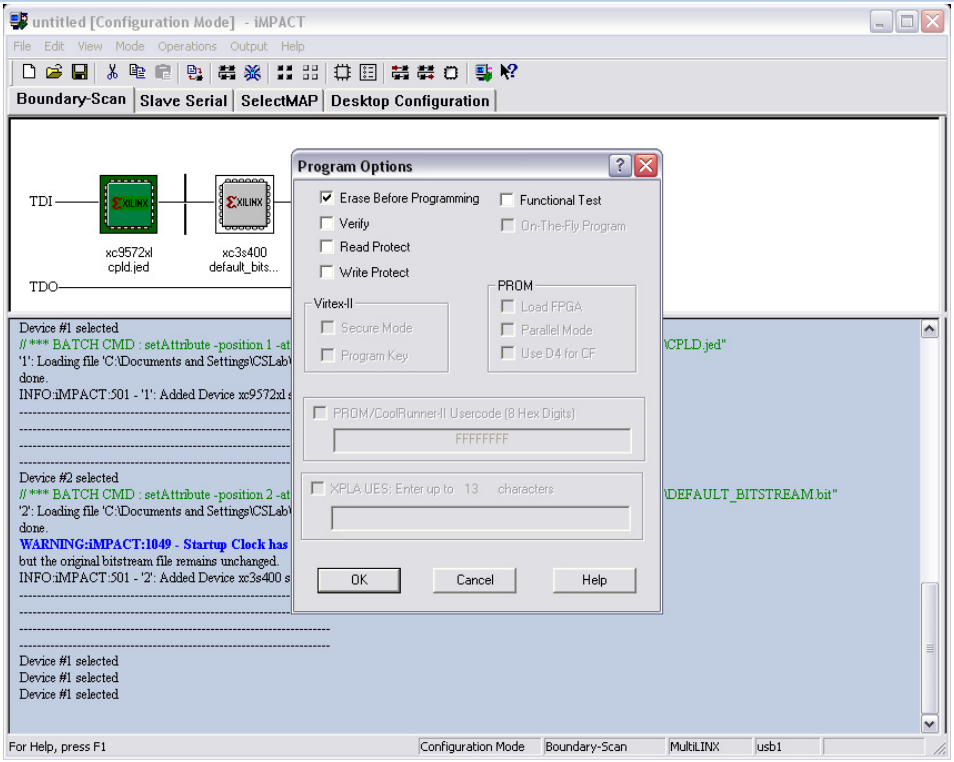


Figura 17 – Program Options para a programação do CPLD

De seguida clique no botão "OK" e dar-se-á inicio à programação do CPLD. Este processo demora uns minutos, aguarde a sua conclusão (Figura 18). Após a mensagem de sucesso o CPLD encontra-se programado com a configuração inicial.

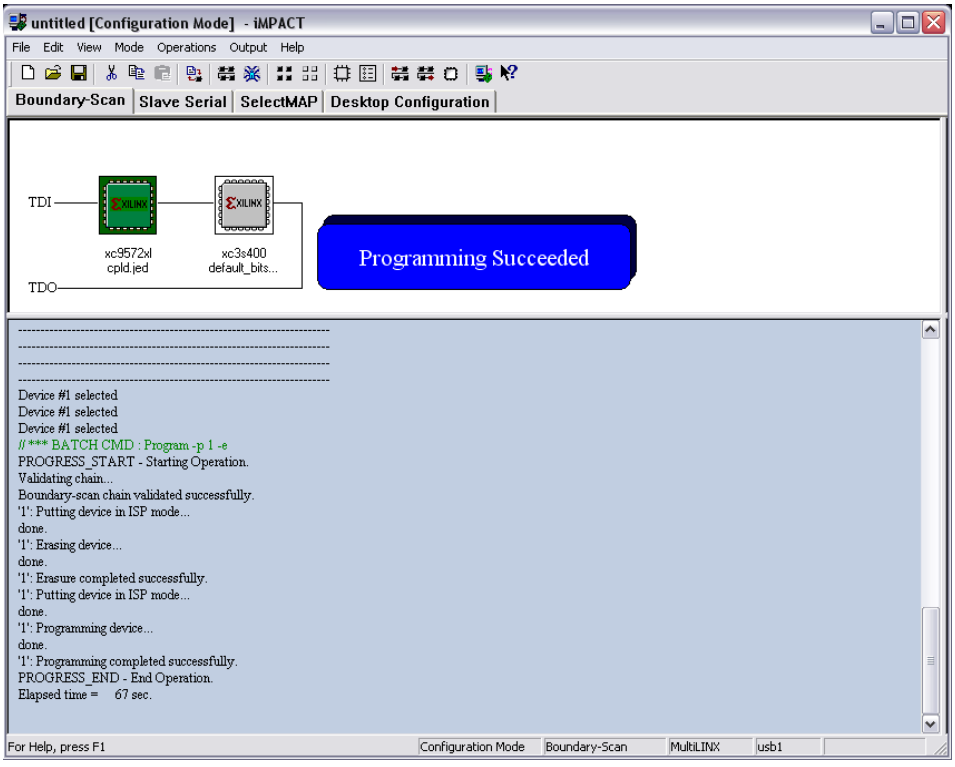


Figura 18 – Mensagem de Sucesso de Programação do CPLD

Se por alguma razão a programação falhar, deve voltar ao inicio do passo 3, até que obtenha a mensagem de sucesso.

Passo 4

- Programação da FPGA

Para a programação da FPGA com a Default BitStream, clique com o botão direito sobre o bloco correspondente à FPGA, e no menu que aparece escolha “program…” (Figura 19)

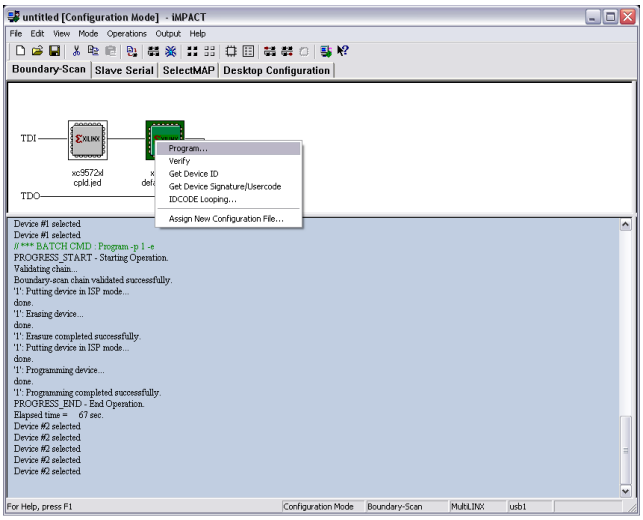


Figura 19 – Menu do botão direito do rato sobre a FPGA na janela do iMPACT Xilinx

Na janela das opções de programação da FPGA, não necessita de seleccionar nenhuma das opções (Figura 20).

ATENÇÃO: ***Antes de prosseguir leia com atenção ao que é pedido!***

Neste momento, é preciso fazer uma ligação física ligando o pino 1 do Oscilador à massa. Para esse efeito faz-se uso do fio auxiliar referido anteriormente na listagem do material. Este deverá ligar no “gnd” (ground) do barramento de alimentação e na outra extremidade fazer contacto com o primeiro pino do Oscilador (Figura 21). Durante todo o processo de transferência de dados e programação da FPGA com a Default BitStream esta ligação não se deve quebrar. Para isso, mantenha fixa a posição com que segura o fio auxiliar. Estando garantida esta ligação, pode prosseguir clicando no botão “OK” e aguarde a mensagem de sucesso (Figura 22).

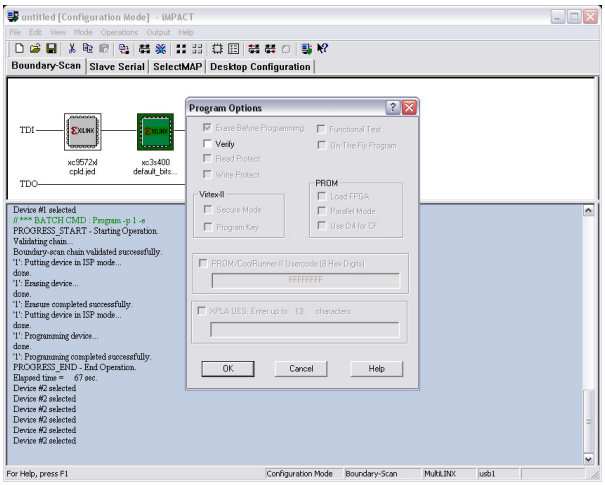


Figura 20 – Program Options para a programação da FPGA

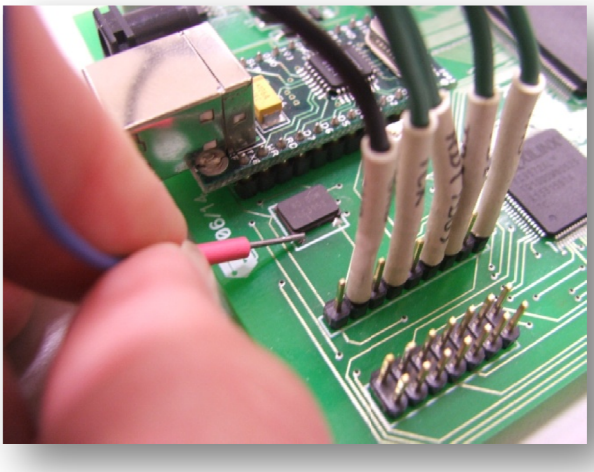


Figura 21 – Exemplificação de como efectuar ligação do pino 1 do oscilador à massa.

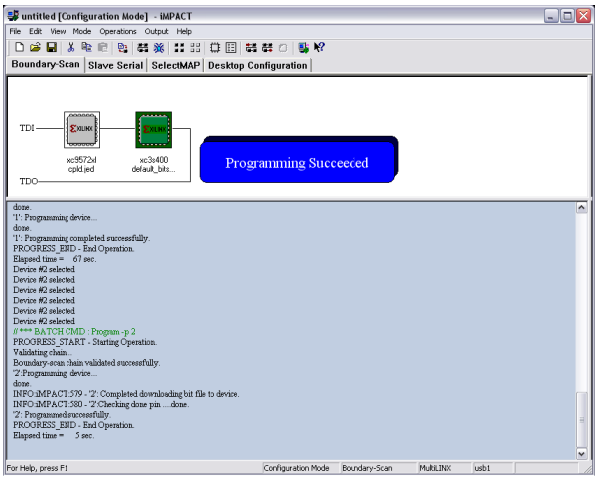


Figura 22 – Mensagem de Sucesso de Programação da FPGA

Caso não tenha conseguido obter sucesso, repita todo passo 4. Se por algum motivo mesmo assim não conseguir, volte ao passo 2 efectuando apenas as tarefas que ainda não conseguiu concretizar. (Ou seja, não precisa de programar novamente o CPLD se já o fez ou de o fazer agora, mesmo que anteriormente não tenha pretendido o fazer).

Obtida a mensagem de sucesso (Figura 22) poderá soltar o fio do pino 1 do Oscilador mas tenha o **cuidado de não o permitir que entre em contacto com qualquer outra parte dos circuito ou pinos da placa.**

Feche a janela do iMPACT e abra a aplicação “Prototyping Board Manager”. (Pode efectuar duplo-clique sobre o atalho existente no Desktop, caso esteja a trabalhar no computador do Lab 0.24 do IEETA).

Clique no menu “Bitstreams → Upload default bitstream…” (Figura 23)

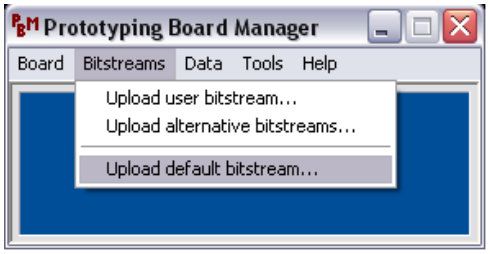


Figura 23 – Prototyping Board Manager

Leia a mensagem de aviso que aparecerá e responda "Yes". Esta apenas pede confirmação que pretende substituir a Default BitStream existente (Figura 24).

É neste momento que o Jumper introduzido no inicio terá efeito, permitindo apagar o primeiro sector que contém a Default BitStream a substituir.

Figura 24 – Aviso do PBM que irá substituir a Default BitStream existente

Na janela de navegação que se segue, escolha o ficheiro "DEFAULT_BITSTREAM.bit". Este é o pretendido para configuração da primeira zona de memória flash. Se estiver utilizando o computado do Lab 0.24, este ficheiro encontra-se no "Desktop" (Figura 25).

Estando seleccionado, clique em "Open".

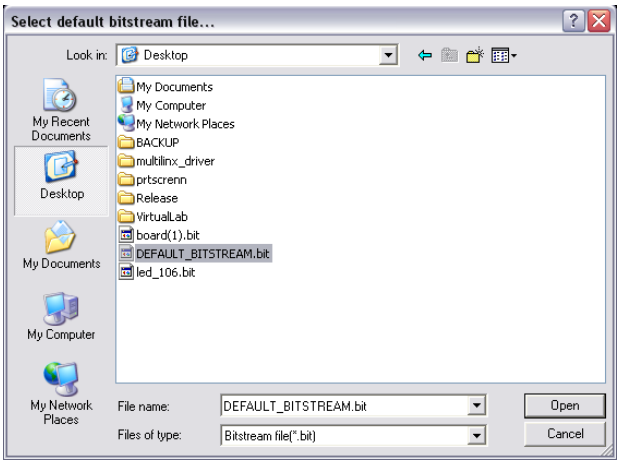


Figura 25 – Selecção do ficheiro .bit contendo a Default BitStream, para transmitir para a primeira zona da memória flash.

A aplicação PBM irá de imediato apagar os sectores de memória correspondentes, e enviar os novos dados.

Se obter uma caixa com a informação “Bitstream upload complete.” (figura 26), então o processo de programação da FPGA terminou com sucesso. Termine clicando em “OK”.

Em caso de obter outro qualquer tipo de mensagem, (como por exemplo dessincronização da placa) deverá tentar desligar, e ligar a placa no interruptor da mesma. E tentar mais uma vez efectuar a transmissão da Default BitStream com o PBM.

Se mesmo assim não conseguir, deverá repetir todo o passo 4.

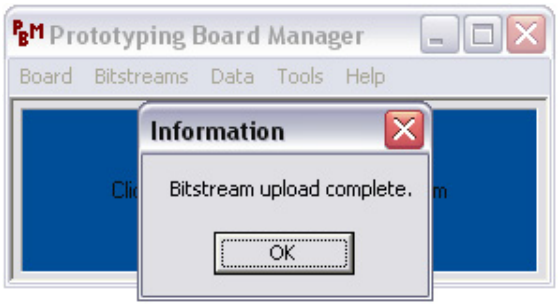


Figura 26 – Informação que o upload da Default BitStream concluiu com sucesso

Neste momento deverá desligar a placa no interruptor On/Off e passar ao passo 5, no qual pode testar a conectividade com a placa e se esta ficou devidamente (re)programada.

Passo 5
- Teste de funcionamento da placa DETIUA-S3

Ligue a placa DETIUA-S3 ao computador através do cabo USB;

Active a placa, através do interruptor On/Off. (On - Luz vermelha acende);

Pressione o botão de pressão “Config” ;

Abra a aplicação “Prototyping Board Manager” (caso já esteja aberta não necessita de abrir novamente um nova instância da aplicação);

Navegue até ao menu “Tools → Terminal window” (Figura 27);

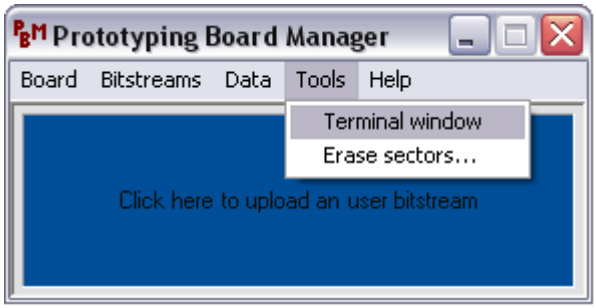


Figura 27 – PBM Tools Menu

Na janela que aparece, no campo “To send:”, introduza a seguinte instrução em ***Hexadecimal***: “5 f f f f f” (deverá introduzir um espaço entre cada caracter à excepção do último – Figura 28).

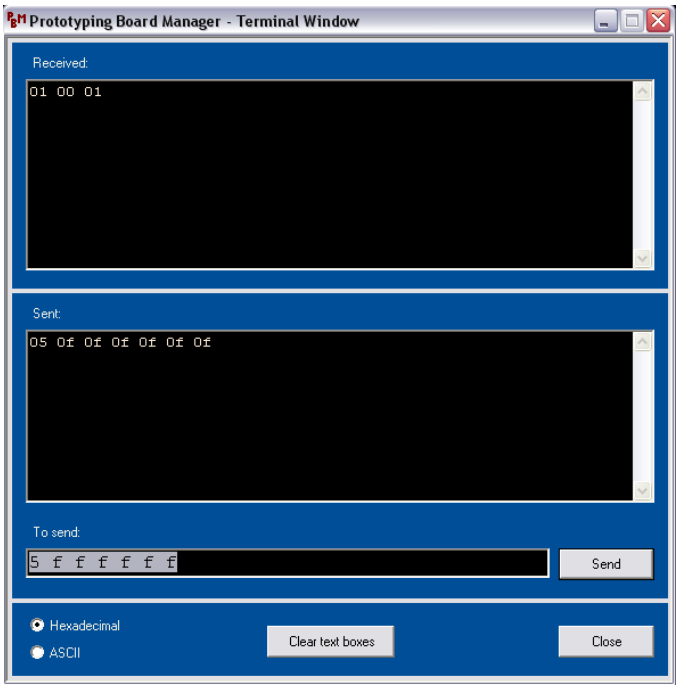


Figura 28 – Terminal window do PBM

Clicando no botão “Send”, o valor introduzido será copiado para a janela inferior “Sent:” e enviado à placa DETIUA-S3.

Se esta possuir as configurações correctas e estiver a funcionar normalmente, responderá ao comando enviado com o valor “01 00 01” ou outro valor de 6 dígitos, que surgirá na janela superior “Received:”.

Se não receber quase de forma instantânea a resposta da placa, então esta possui uma anomalia à qual poderá ser ou não solucionada com a (re)programação da FPGA ou se for caso, do CPLD. Tente novamente os passos 1 a 5 desde tutorial. Se mesmo assim o problema persiste informe o docente responsável pela placa, ou responsável pelo material do Lab 0.24 do IEETA expondo a situação de modo a que este possa ajudar de forma mais precisa alguma situação não contemplada neste documento ou outros referidos na bibliografia.

Bibliografia

Para elaboração deste tutorial foram consultadas as seguintes obras, documentos ou sites online:

[1] Manual da placa DETIUA-S3:

- http://www.ieeta.pt/~skl/Research/Projects/Manual_Utilizador_pt.pdf

[2] Site online da Xilinx:

- http://www.xilinx.com/products/design_tools/logic_design/design_entry/impact.htm
- http://toolbox.xilinx.com/docsan/3_1i/data/common/hug/hug.htm
- http://www.xilinx.com/support/answers/8097.htm
- http://www.xilinx.com/support/documentation/customer_notices/xcn05010.pdf
- http://toolbox.xilinx.com/docsan/xilinx4/data/docs/pac/pac.html

[3] Manual de utilização do Prototyping Board Manager (PBM):

- http://www.ieeta.pt/~skl/Research/Projects/PBM_User_Manual_PT.pdf

[4] Drivers para cabo USB Xilinx MultilinX :

- ftp://ftp.xilinx.com/pub/applications/misc/mltlnx_ans_14782.zip

[5] Drivers para placa DETIUA-S3:

- http://www.ieeta.pt/~skl/CR/DITIUAS3/d10606.rar

4. Manual de Utilização do iCmips 1.0

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática

Manual de Utilização

Versão 1.0

Manual explicativo da configuração e utilização do software iCmips e hardware complementar, desenvolvido para interagir com um processador de arquitectura MIPS descrito em VHDL e implementado na placa DETIUA-S3.

Bernardo Francisco Peralta Pires da Silva
Bernardo.Silva@ua.pt
Setembro de 2008

Índice

Introdução

O software iCmips 1.0 foi desenvolvido em 2007/08 no âmbito da dissertação do aluno Bernardo Silva em Mestrado Integrado em Engenharia de Computadores e Telemática na Universidade de Aveiro, na qual se desenvolveu um processador em VHDL de arquitectura MIPS para ensino. Após alguma análise, sentiu-se a necessidade de possuir uma ferramenta capaz de facilitar a execução de testes ao processador implementado, assim como facilitar a compreensão de valores obtidos à saída dos módulos constituintes da arquitectura implementada.

Assim surgiu o "Eu vejo o MIPS" (do trocadilho da leitura oral em inglês da expressão: "I C *(see)* MIPS"), *iCmips,* uma ferramenta de software complementar a um protocolo de comunicação desenvolvido e aplicado na placa de prototipagem DETIUA-S3 da Universidade de Aveiro que dá a possibilidade de o utilizador comunicar com um processador de arquitectura MIPS em tempo real, verificando a sua integridade e autenticidade dos resultados produzidos.

Ao contrário do que possa parecer, este software não é mais um simulador de processadores de arquitectura MIPS, mas sim um software capaz de comunicar com hardware (FPGA) que possui implementado fisicamente o processador que se encontra em execução e produzindo resultados reais.

Neste manual o utilizador poderá familiarizar-se com a ferramenta iCmips 1.0, conhecendo as suas características, limitações e vantagens, assim como descobrir o funcionamento do protocolo de comunicação implementado com o qual o software comunica.

No final da leitura deste manual, o leitor deverá ser conhecedor dos mecanismos e ferramentas que possui à sua disposição para que possa desenvolver e testar novos processadores de arquitectura MIPS com a placa de prototipagem DETIUA-S3.

Requisitos do Sistema

Este software foi desenvolvido para trabalhar especificamente com a placa de prototipagem DETIUA-S3. Para que o software funcione correctamente é necessário reunir as seguintes características:

De Hardware:

- Placa de Prototipagem DETIUA-S3
- Cabo USB do tipo A-B
- Computador/Portátil com pelo menos 1 interface USB (Universal Serial Bus) versão ≥ 1.0

De Software:

- Sistema operativo Microsoft Windows 2000, XP ou Vista
- Ter instalado os drivers FTD2XX para reconhecimento da placa DETIUA-S3 [4]
- Possuir instalado o software .NET Microsoft Framework 2.0 ou versão superior [5]

  NOTA: Microsoft Windows Vista já possui instalado por defeito a Framework 3.0 razão pela qual já não precisa de instalar mais nenhuma versão para execução correcta do software.
- Xilinx ISE 9.2 ou versão superior (***opcional*** - necessário apenas para editar/desenvolver projectos)

Nota1: A versão de software 1.0 do iCmips foi testada em diversos sistemas operativos, funcionando garantidamente em Microsoft Windows 2000, XP e Vista. Não existe garantias, nem suporte para que este software funcione em ambientes Unix, MAC OS ou outros.

Nota2: O instalador do software iCmips 1.0 detecta se possui ou não instalado a .NET Framework 2.0 ou superior. Se não possuir, automaticamente o programa pede-lhe se pretende instalar a versão .NET Framework 2.0, efectuando o download da internet do software.

Configuração e Uso do Software

Instalação do iCmips 1.0

Para instalar o programa iCmips num computador deve possuir o ficheiro correspondente ao programa de instalação "Setup iCmips 1.0 Students.msi" e "Setup.exe". Estes ficheiros consistem num instalador interactivo de fácil uso para o utilizador que cria as pastas, ficheiros e atalhos para esta distribuição de software. Durante a instalação leia tudo com cuidado e atenção.

Passo 1

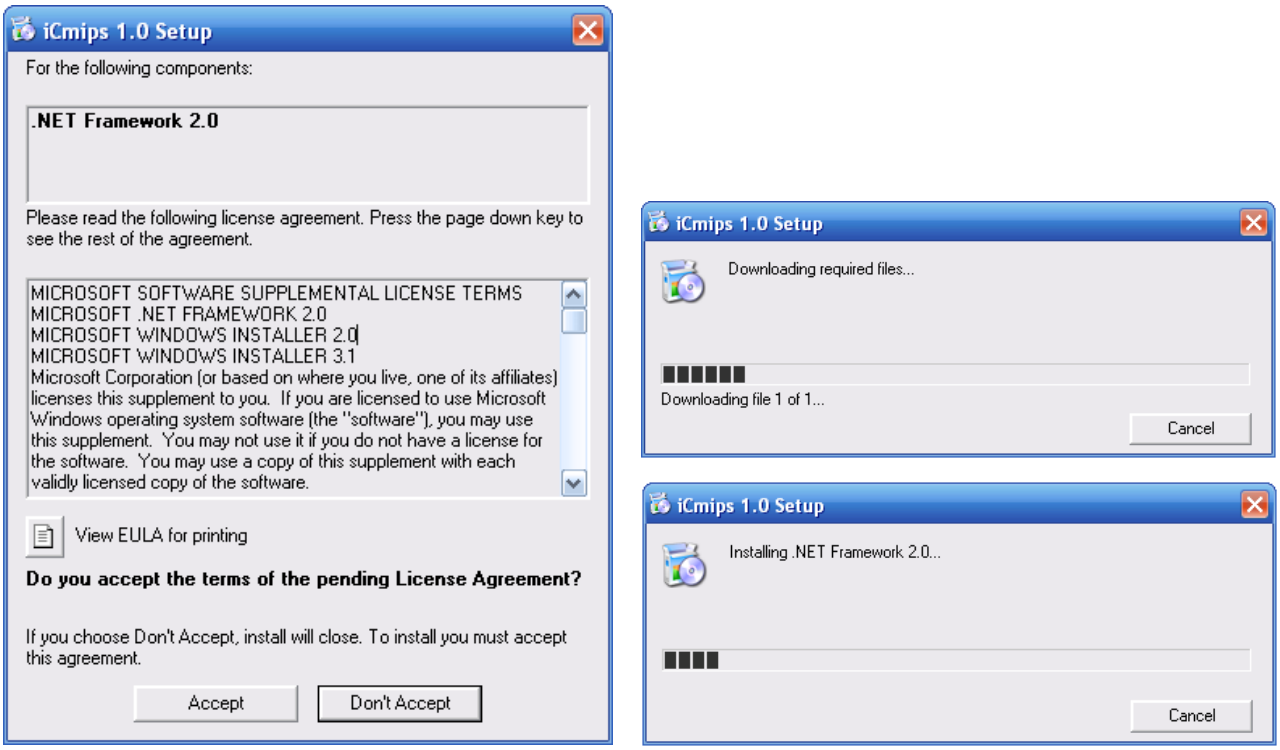


Figura 1 – Janelas de instalação do .NET Framework 2.0.
À esquerda janela de inicial da instalação,
à direita em cima a janela de download do .NET Framework 2.0 e
em baixo a sua instalação.

Para iniciar a instalação do iCmips execute o ficheiro "Setup.exe".

Se possuir o .NET Framework instalado, passe para o próximo passo. Senão verá a janela inicial de instalação da plataforma .NET Framework 2.0 (Figura 1 á esquerda). Carregue no botão "Accept" para continuar a instalação. Neste momento precisa de obter ligação à internet pois será feito o download da Framework (Figura 1 à direita). Aguarde uns minutos até lhe aparecer a janela de instalação do iCmips (ver passo 2).

Passo 2

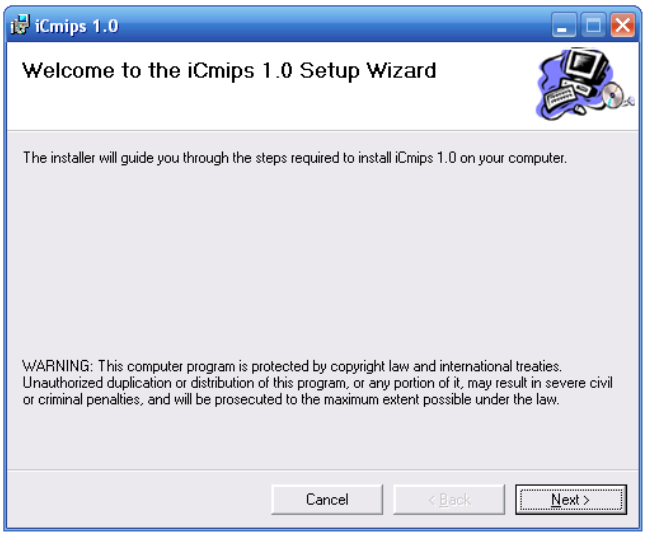


Figura 2 – Janela de bem-vindo ao programa de instalação do iCmips 1.0

Ao ver a janela da Figura 2, o utilizador já possui o .NET Framework instalado no seu computador e pode dar inicio à instalação do software iCmips 1.0.

Clique no botão "*Next >*" para continuar a instalação.

Passo 3

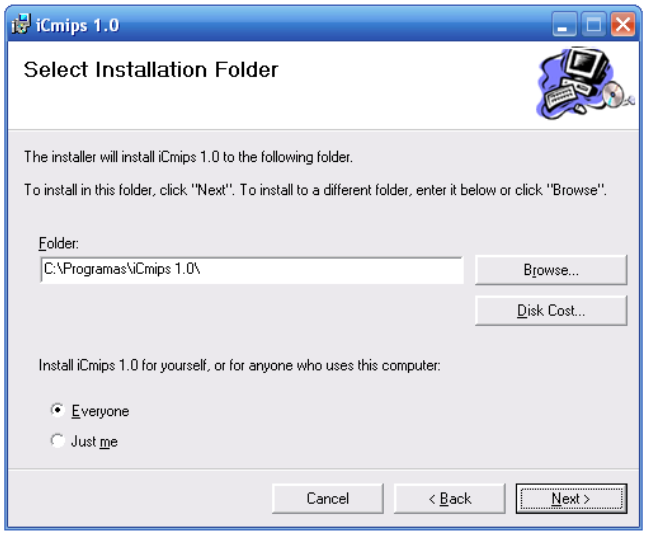


Figura 3 - Janela de escolha do directório de instalação do software iCmips 1.0

Neste passo o utilizador é livre de escolher o directório aonde ficará instalados os ficheiros do iCmips 1.0 (ver Figura 3). Por defeito o programa será instalado em "C:**%directório de programas%**\iCmips" e serão instalados atalhos no ambiente de trabalho e no menu Iniciar -> Programas -> iCmips 1.0. Para mudar o directório clique no botão "Browse…" e escolha a localização de instalação pretendida. Se pretender saber quanto espaço ocupa a instalação do iCmips 1.0 clique no botão "Disk Cost…".

Passe para o próximo passo de instalação clicando em *"Next >"*.

Passo 6

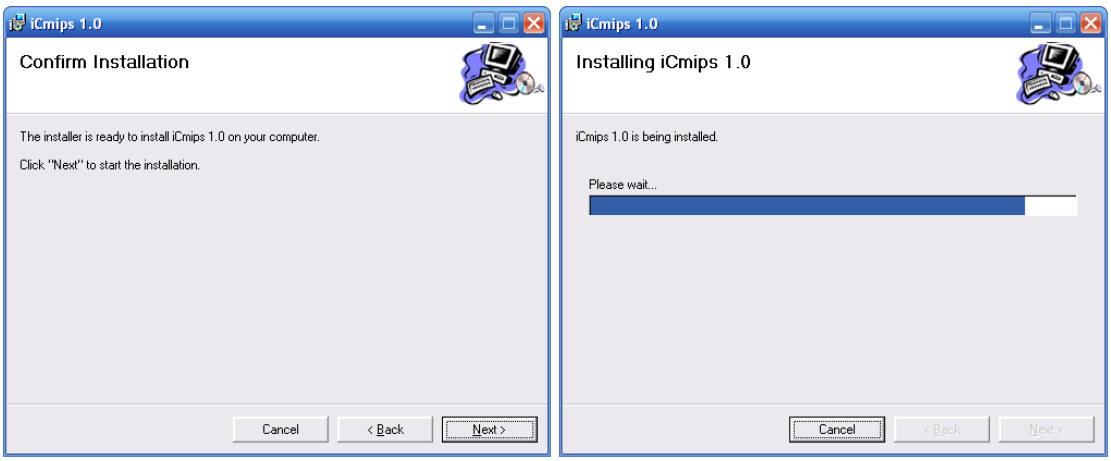


Figura 4.a (à esquerda) e 4.b (à direita) – À esquerda a janela de confirmação e à direita a janela de progressão da instalação do iCmips 1.0

Neste momento estão reunidas todas as condições para instalação do software iCmips 1.0. Pressione o botão *"Next>"* (ver Figura 4.a) para dar inicio à instalação do programa no seu computador.

Visualizará uma nova janela do programa de instalação, em poderá acompanhar a progressão da instalação (ver Figura 4.b).

Passo 7

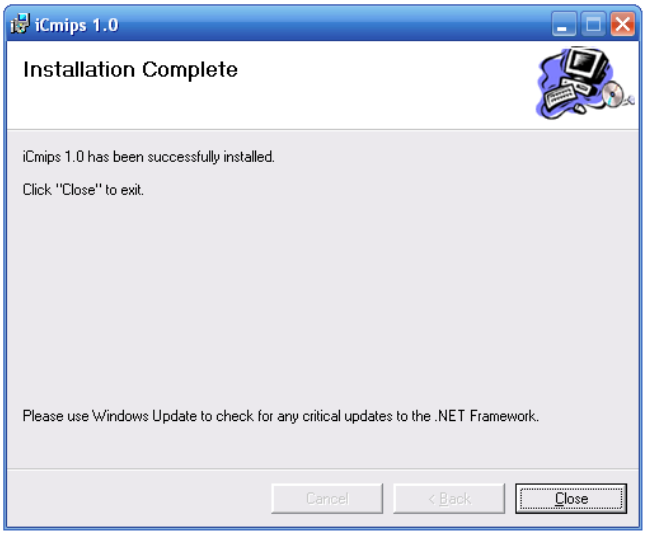


Figura 5 – Janela final da instalação do iCmips 1.0

Neste momento o software iCmips 1.0 encontra-se instalado no seu computador (ver Figura 5). Para abrir o programa, encontra um atalho no ambiente de trabalho ou pode encontrar no menu Iniciar→ Programas → iCmips 1.0 o atalho de iniciação do programa e outros links entre os quais: para este manual na língua portuguesa, um para o mesmo manual mas na língua inglesa e um que abre o projecto base no programa Xilinx ISE caso possuir instalado.

Para concluir e fechar o programa de instalação, prima o botão *"Close"*.

A arquitectura do programa

O programa iCmips consiste numa ferramenta de controlo e visualização de dados de um processador MIPS. A janela principal do iCmips é pode ser dividida em quatro áreas distintas (ver Figura 6 – A, B, C e D):

A – Barra de Menus;
B – Barra de Ferramentas;
C – Sub-Janelas de visualização de dados;
D – Barra de Estado e Informações.

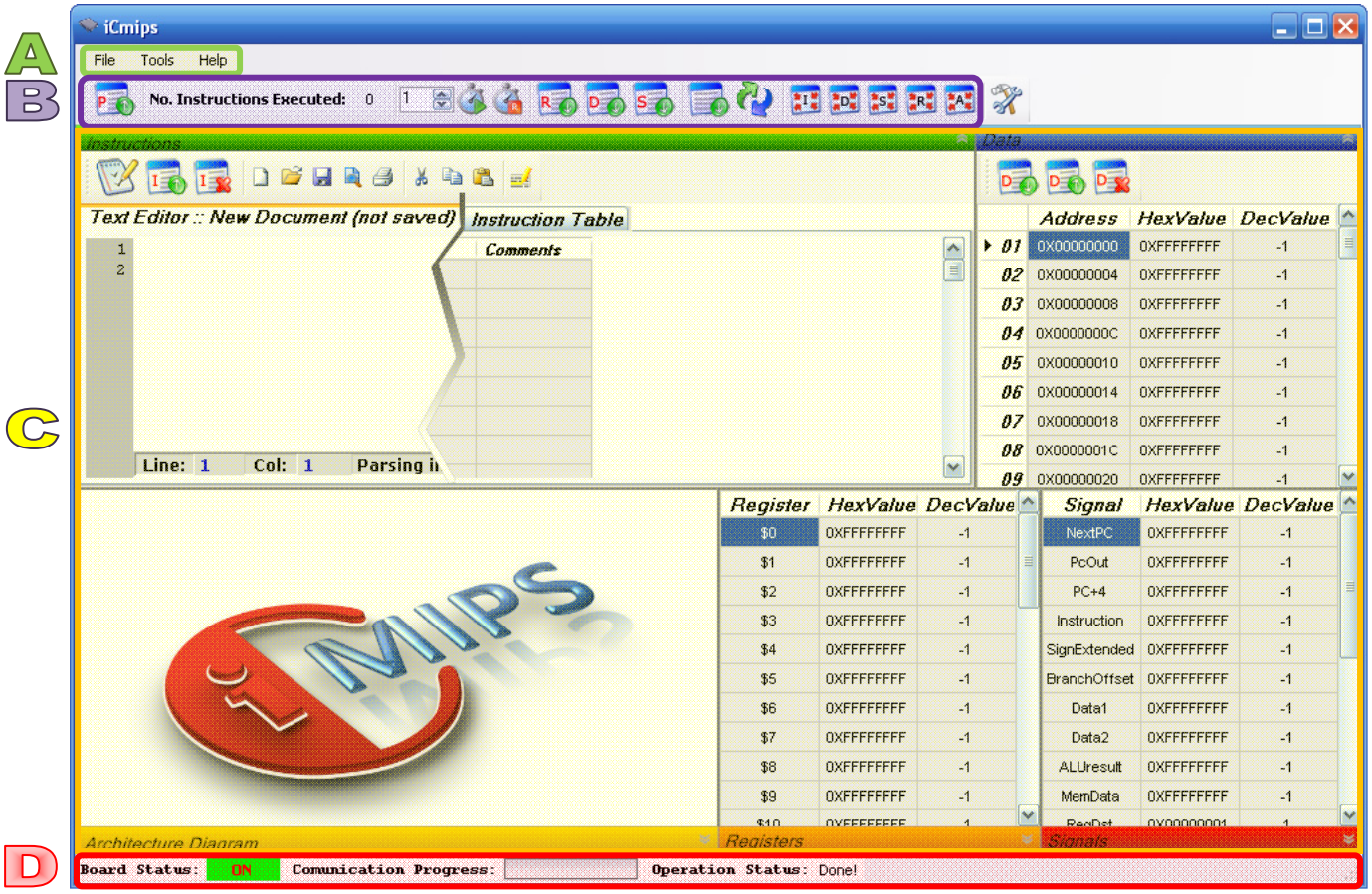
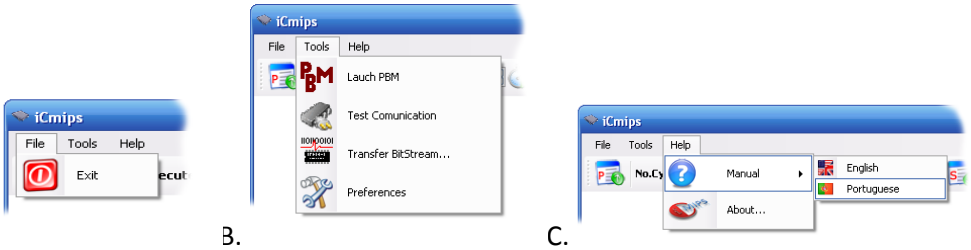


Figura 6 – Janela do iCmips 1.0 e identificação das varias áreas de trabalho

De seguida nos próximos subcapítulos poderá encontrar informação mais específica relativa a cada uma das áreas anteriormente mencionadas, de modo a que possa ficar a conhecer aonde e como obter os resultados pretendidos.

Barra de…

Menus



A. B. C.

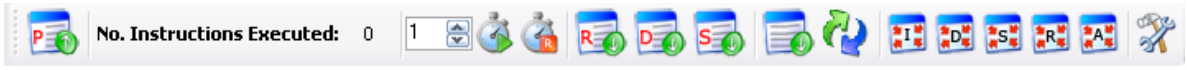
Figuras 7 – Os submenus da barra de menus: A. Menu ficheiro (*"File"*); B. Menu ferramentas (*"Tools"*); C. Menu de Ajuda (*"Help"*).

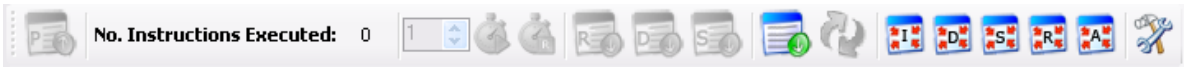
Navegando pela barra de menus o utilizador tem a possibilidade de efectuar as seguintes operações:

Opção	*Acção*
Exit	Termina a execução do programa iCmips.
Launch PBM	Executa o programa PBM caso este se encontre instalado no computador.
Test Connection	Efectua um teste de comunicação com a placa DETIUA-S3, verificando a presença , conectividade e a existência do protocolo correcto de comunicação na placa.
Transfer BitStream…	Esta funcionalidade permite-lhe enviar uma user bitstream com a qual deseja configurar a FPGA. Ao escolher um ficheiro válido, a bitstream é enviada para a placa e armazenada na user bitstream area da memória flash. Depois desta operação, a bitstream permanece disponível para configurar a FPGA até transferir outra user bitstream.
Preferences	Esta opção abrirá a janela de preferências do iCmips aonde na qual o utilizador poderá efectuar as configurações necessárias para ajustar os parâmetros do iCmips à arquitectura de processador MIPS desenvolvido e algumas configurações de personalização do programa.
Help	Neste menu o utilizador pode abrir a versão electrónica deste manual que se encontra disponível em português ( ) ou em inglês ( ).
About…	Esta funcionalidade apresenta alguma informação acerca do iCmips 1.0.

Tabela 1 – Descrição dos botões e respectivas operações da barra de menus

Ferramentas

A. 

B. 

Figuras 8 – Barra de Ferramentas
A) Estado da barra de ferramentas quando a placa DETIUA-S3 encontra-se ligada
B) Estado da barra de ferramentas quando a placa DETIUA-S3 encontra-se desligada

É na barra de ferramentas que o utilizador vai encontrar a maior parte dos botões para interacção com o programa e com a placa DETIUA-S3. Todos os botões que estejam associados com qualquer tipo de operação que envolva comunicação com a placa, caso esta se encontre desligada, os botões encontra-se inactivos (Figura 8-A). Assim que o software auto-detecte a placa DETIUA-S3, os botões voltam a estar activos.

Na tabela seguinte (Tabela 2) encontra uma breve descrição de cada botão da barra de ferramentas e qual o seu efeito quando pressionado.

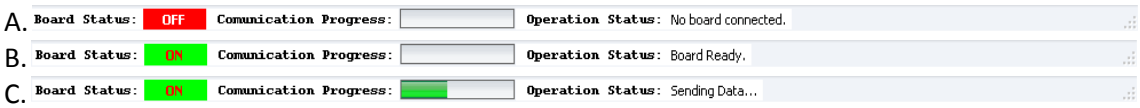
Opção	*Acção*
 **Program**	Quando pressionado o iCmips irá tentar transferir o conteúdo das tabelas nas sub-janelas Instruções e Dados para a memória de instruções e dados, repectivamente.
No.Instructions Executed: #	Texto informativo indicando o número de instruções executadas desde o último reset efectuado ao processador.
 **No. of Instructions**	Número de instruções a serem geradas no processador MIPS. Este valor pode variar entre 1 e 100 instruções.
 **Generate Clock Cycle**	Quando pressionado é gerado ciclos de relógio no processador, tantos quanto o valor seleccionado em ***No. of Instructions***, actualizando o campo "No.Instructions Executed".
 **Generate Reset Cycle**	Quando pressionado é activado o estado de reset do processador e gerado ciclos de relógio no processador, tantos quanto o valor seleccionado em ***No. of Instructions***, actualizando o campo "No.Cycles Executed" para o valor zero.
 **Read Registers**	Esta funcionalidade permite ao utilizador efectuar a leitura de todos os registos do processador MIPS para a tabela da sub-janela Registos (Registers). A quantidade de registos lidos e porta de leitura é configurada na janela de Preferências.

 **Read Data Memory**	Esta funcionalidade permite ao utilizador efectuar a leitura de todo o conteúdo da memória de dados do processador MIPS para a tabela da sub-janela Dados (Data). A dimensão da memória de dados e a porta de leitura é configurável na janela de Preferências.
 **Read Signals**	Esta funcionalidade permite ao utilizador efectuar a leitura de todos os sinais que foram ligados às portas de entrada de sinais e visualizar o seu valor na tabela da sub-janela de sinais (Signals). A configuração e descrição da dimensão dos sinais é efectuada na janela de Preferências.
 **Auto get data(ON|OFF)**	Este botão permite ao utilizador ligar ou desligar a leitura automática dos registos, memória de dados e sinais após um comando de geração de ciclos de relógio ou reset. Por defeito esta opção encontra-se activa (imagem da esquerda com seta a verde), no entanto pode ser desligada (imagem da direita com seta a vermelho).
 **Synchronize Board**	Durante a comunicação com a placa DETIUA-S3 podem surgir erros de fluxo de informação, "suspendendo" o funcionamento da placa (protocolo encontra-se no estado 2 da tabela 4). Esta funcionalidade vem dar solução a essa situação procurando sincronizar novamente a placa com o iCmips sem necessidade de disparar a reconfiguração da FPGA e perda dos ciclos de relógio já efectuados.
 **View/Hide Instructions subwindow**	Esta funcionalidade permite a encolher ou expandir a sub-janela de Instruções (Instructions).
 **View/Hide Data subwindow**	Esta funcionalidade permite a encolher ou expandir a sub-janela de Dados (Data).
 **View/Hide Signals subwindow**	Esta funcionalidade permite a encolher ou expandir a sub-janela de Sinais (Signals).
 **View/Hide Registers subwindow**	Esta funcionalidade permite a encolher ou expandir a sub-janela de Registos (Registers).
 **View/Hide Architecture subwindow**	Esta funcionalidade permite a encolher ou expandir a sub-janela do Diagrama da Arquitectura implementada (Architecture Diagram).
 **Edit/View Preferences**	Esta funcionalidade permite abrir a janela de Preferências e Configurações do programa iCmips.

Tabela 2 – Descrição dos botões e respectivas operações da barra de ferramentas

Esta barra de ferramentas tem a capacidade de receber outros botões que se encontram localizados nas sub-janelas, assim como de deslocação para a área de sub-janelas e vice-versa. Apesar destas capacidades, a disposição espacial dos botões das sub-janelas e da barra de ferramentas no programa não é conservada entre sucessivas utilizações do programa, volta à configuração por defeito em cada execução do iCmips 1.0.

Estados



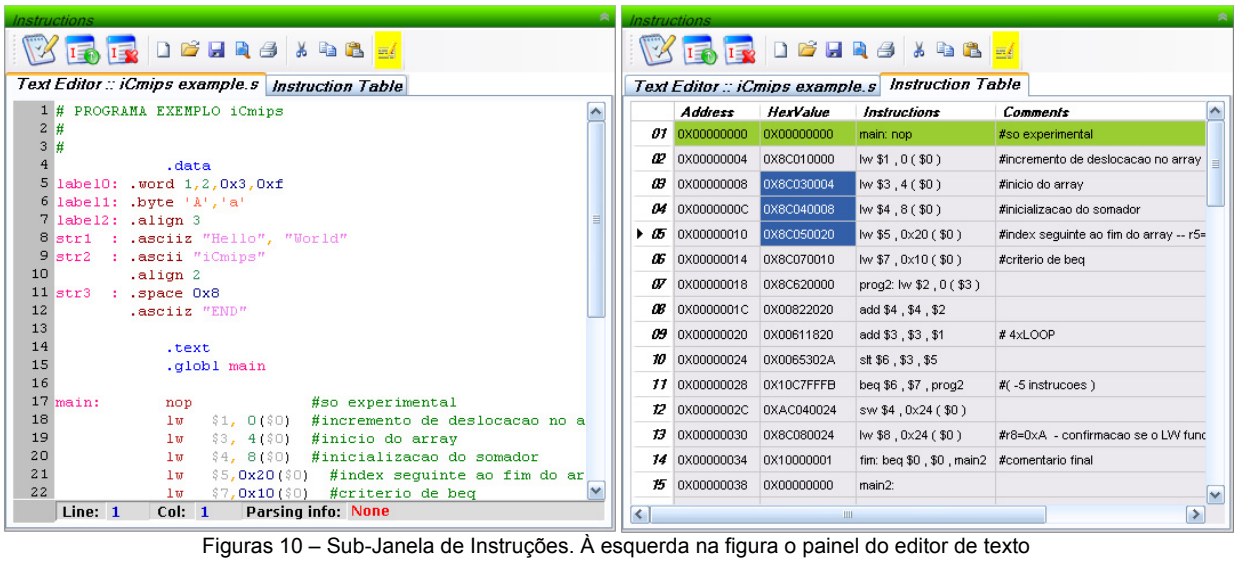
Figuras 9 – A barra de estado quando:
A. a placa DETIUA-S3 encontra-se desligada;
B. a placa DETIUA-S3 encontra-se ligada e foi auto-detectada;
C. o programa iCmips encontra-se em comunicação com a placa DETIUA-S3

A barra de estados do programa iCmips (Figura 9) dá a conhecer ao utilizador três informações relativas à placa DETIUA-S3: se placa encontra-se ligada, o progresso de uma operação com a placa e que operação se encontra em execução. Durante a execução de qualquer operação com a placa o utilizador pode manter-se actualizado com o estado da placa observando as informações desta barra.

Área de Sub-janelas

A área de sub-janelas de visualização de dados é constituída por cinco sub-janelas. Cada uma dessas sub-janelas é específica a um tipo de dados do processador: dados de instruções, dados guardados na memória de dados, dados dos registos, valores dos sinais e diagrama da arquitectura do processador MIPS implementado.

Sub-janela de Instruções



Figuras 10 – Sub-Janela de Instruções. À esquerda na figura o painel do editor de texto e à direita o painel da tabela de instruções.

A sub-janela de instruções permite ao utilizador efectuar duas tarefas importantes: abrir/editar/guardar ficheiros mips assembly (Figura 10 à esquerda) e efectuar uma análise semântica e sintáctica das instruções assembly convertendo-as em código binário para uma tabela (Figura 10 à direita) com possibilidade de posteriormente transferir as instruções para a memória de instruções do processador implementado na placa DETIUA-S3.

No topo desta sub-janela uma barra de ferramentas contendo operações relativas ao conteúdo desta janela. A tabela seguinte (Tabela 3) explícita o significado de cada botão:

Opção	*Acção*
 **Parse Text**	Este botão possui a função de efectuar uma verificação semântica e sintáctica do código assembly escrito no editor de texto. Na situação de encontrar erros semânticos ou sintácticos, uma mensagem de erro será escrita na barra de informações do editor de texto. Caso a verificação do código assembly seja bem sucedida, será efectuado a conversão das instruções assembly para código máquina e preenchida a tabela de instruções. (Para configuração das instruções suportadas consulte a configuração de instruções na janela de Preferências).
 **Upload Instruction**	Quando pressionado, o programa iCmips iniciará a transferência do código máquina de instruções para a memória de instruções do processador.
 **Erase Instruction Memory**	Quando pressionado, o programa iCmips procederá à escrita completa de toda a memória de instruções com Zeros.
 **New assembly file**	Possibilita a criação de um novo ficheiro assembly. Se possuir um ficheiro já aberto ou texto escrito, será questionado se o pretende guardar antes prosseguir.
 **Open assembly file**	Quando clicado possibilita abrir um ficheiro mips assembly existente (extensão do ficheiro: ".s").
 **Save assembly file**	Permite ao utilizador guardar o texto mips assembly produzido numa localização do seu computador para posterior edição ou uso. O ficheiro é guardado com uma extensão ".s".
 **Print Preview**	Possibilita ao utilizador visualizar uma pré-visualização do aspecto final de impressão em papel do texto existente no editor de texto.
 **Print**	Envia o código assembly existente no editor de texto para o dispositivo de impressão em papel (ou outro definido pelo sistema).
 **Cut Text**	Corta o texto seleccionado no editor de texto e coloca-o no clipboard para posterior utilização.
 **Copy Text**	Copia o texto seleccionado no editor de texto e coloca-o no clipboard para posterior utilização.
 **Paste Text**	Retira o texto do clipboard e insere-o na posição actual do cursor do editor de texto.
 **Highlight Text**	Esta funcionalidade permite ao utilizador activar a coloração do código assembly. Desta forma o utilizador poderá visualizar de forma mais fácil a sintaxe assembly no editor de texto.

Tabela 3 – Descrição das operações permitidas na barra de ferramentas da sub-janela de instruções

Para que o utilizador possa efectuar operações sobre a memória de instruções é necessário que seja integrado o módulo "SetInstMEM.vhd" no projecto implementado. Verifique o capítulo de Configuração e Uso do Hardware neste manual para mais informações de como efectuar a implementação correcta deste módulo.

O editor de texto da aplicação é equipado com um mecanismo de verificação semântica e sintáctica da linguagem MIPS assembly. Como todas as linguagens de programação também a linguagem assembly possui variadas directivas e declarações possíveis de ser utilizadas. Nesta versão do software iCmips 1.0 foram apenas admitidas as seguintes características da sintaxe da linguagem assembly:

- A directiva ".Data" é estritamente obrigatória antes de declarar outras directivas para armazenamento de valores, arrays de dados, alinhamentos ou reservas de memória;
- As directivas admitidas nesta versão são: .word, .byte, .align, .ascii, .asciiz e .space;
- Seguida à directiva ".Text" deve-se encontrar a ".globl" seguida do nome da label que identifica o inicio do programa assembly;
- Não é necessário que a directiva ".Data" seja declarada, assim como não é obrigatório que seja declarada antes da directiva ".Text";
- A memória de dados é preenchida sequencialmente conforme as directivas encontradas;
- A memória de instruções é preenchida sequencialmente a partir da label que identifica o inicio do programa assembly;
- Todas as instruções suportadas encontram-se declaradas na janela de Preferências sendo sujeitas a activação, passíveis de modificação ou introdução de novas instruções para que seja possível a análise semântica e sintáctica do código assembly e geração do código máquina.
- O programa suporta labels para controlo de fluxo da execução mas, apesar de puderem ser declaradas, as posições de memória de dados referentes às labels de declaração de dados não são calculadas.

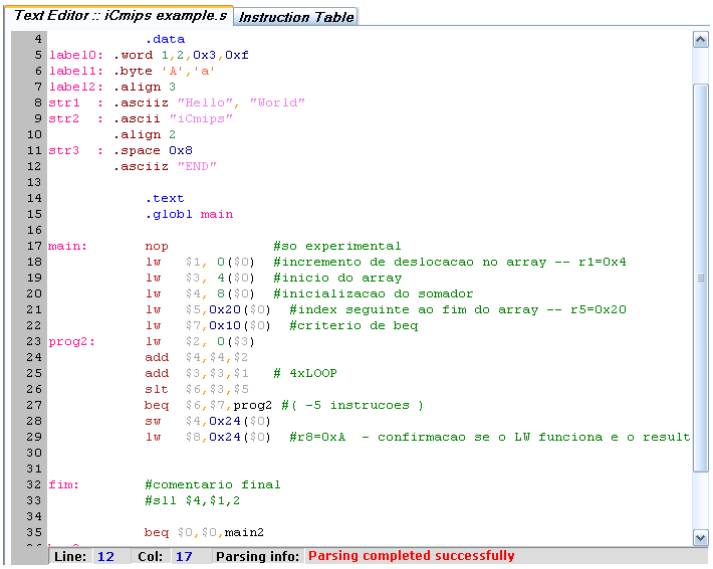


Figura 11 – Exemplo de um programa escrito em assembly

A dimensão da tabela de instruções é definida segundo o número de words de 32bit que a memória de instruções implementada suporta. Veja "Instruction Memory Size" nas configurações do processador do utilizador da janela de Preferências. A má especificação da dimensão da memória poderá levar à escrita incorrecta dos dados nos endereços de memória.

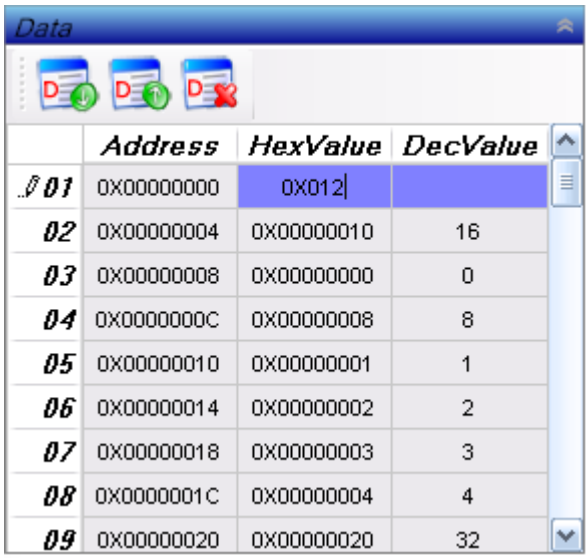
O utilizador pode encontrar já alguns programas exemplo na pasta "Assembly" que se encontra no directório de instalação do iCmips. Cada ficheiro possui um programa de demonstração para processadores já sintetizados e prontos para transferir para a placa DETIUA-S3.

Sub-janela de Dados

A sub-janela de dados permite ao utilizador ler e escrever o conteúdo da memória de dados de um processador MIPS e visualizar os valores numa tabela.

Quando é efectuado um parser sobre um código fonte assembly e este possua declarações de dados (.word, .byte, .asciiz, etc..) na secção *".Data"* esses dados serão carregados na tabela desta sub-janela para posterior escrita na memória de dados (ver exemplo da Figura 12). No entanto o utilizador pode efectuar a inserção directa na tabela dos valores que pretende armazenar na memória do processador. Para isso seleccione a inserção em hexadecimal (ou decimal) junto ao endereço de memória que pretende armazenar (veja o exemplo de inserção do valor "0x012" da figura 12).

A sub-janela de dados é dotada ainda de uma barra de ferramentas com 3 botões com função de leitura, escrita e apagar a memória de dados (no topo da Figura 12, da esquerda para a direita, respectivamente).



	Address	HexValue	DecValue
01	0X00000000	0X012	
02	0X00000004	0X00000010	16
03	0X00000008	0X00000000	0
04	0X0000000C	0X00000008	8
05	0X00000010	0X00000001	1
06	0X00000014	0X00000002	2
07	0X00000018	0X00000003	3
08	0X0000001C	0X00000004	4
09	0X00000020	0X00000020	32

Figura 12 – Sub-Janela de Dados (Data)

A dimensão da tabela de dados é definida segundo o número de words de 32bit que a memória de dados implementada suporta. Veja "Data Memory Size" nas configurações do processador do utilizador da janela de Preferências. A má especificação da dimensão da memória poderá levar à escrita incorrecta dos dados nos endereços de memória.

Para que o utilizador possa efectuar operações sobre a memória de dados é necessário que seja integrado o módulo "SetDataMEM.vhd" no projecto implementado. Verifique o capítulo de Configuração e Uso do Hardware neste manual para mais informações de como efectuar a implementação correcta deste módulo. É necessária ainda a correcta configuração da porta de leitura dos valores da memória de dados na janela de Preferências no painel de configuração de portas.

Sub-janela de Registos

Register	HexValue	DecValue
$0	0X00000000	0
$1	0X00000004	4
$2	0X00000001	1
$3	0X00000014	20
$4	0X00000001	1
$5	0X00000020	32
$6	0X00000001	1
$7	0X00000001	1
$8	0X00000000	0
$9	0X00000000	0
$10	0X00000000	0
$11	0X00000000	0
$12	0X00000000	0
$13	0X00000000	0
$14	0X00000000	0
$15	0X00000000	0

Registers

Figura 13 – Sub-Janela de Registos (Registers)

A sub-janela de registos (Figura 13) dá a possibilidade ao utilizador de visualizar o conteúdo do banco de registos do processador implementado.

Para que o utilizador consiga obter os valores para esta sub-janela é necessário que configure o software devidamente, identificando a porta de leitura de valores do banco de registos. Para isso verifique a porta escolhida no painel de configurações de portas na janela de Preferências.

Para que o utilizador possa efectuar leituras de todo o banco de registos é ainda necessário que seja integrado o módulo "SetRegister.vhd" no projecto implementado. Verifique o capítulo de Configuração e Uso do Hardware neste manual para mais informações de como efectuar a implementação correcta deste módulo.

Sub-janela de Sinais

O programa iCmips analisa e processa o valor oriundo dos sinais ligados a 16 portas de 32bit cada. O utilizador tem a possibilidade de poder declarar quais os sinais ligados a que portas e visualizar o valor dos sinais após de cada iteração do processador MIPS (ver janela de Preferências → painel de configuração de portas).

Na sub-janela de sinais, o utilizador pode observar o conteúdo dos sinais oriundos do processador (Figura 14) e confrontá-los com os resultados esperados/pretendidos.

Assim como em todas as outras sub-janelas que apresentem valores de sinais em tabelas, também nesta sub-janela o utilizador possui a apresentação do valor do sinal em base hexadecimal e decimal e ainda com a possibilidade de seleccionar e copiar os valores para outras localizações utilizando as teclas de atalho *Control+C*.

Signal	HexValue	DecValue
NextPC	0X00000004	4
PcOut	0X00000000	0
PC+4	0X00000004	4
Instruction	0X00000000	0
SignExtended	0X00000000	0
BranchOffset	0X00000000	0
Data1	0X00000000	0
Data2	0X00000000	0
ALUresult	0X00000000	0
MemData	0X00000000	0
RegDst	0X00000001	1
Branch	0X00000000	0
MemRead	0X00000000	0
MemToReg	0X00000000	0
ALUOp	0X00000002	2
MemWrite	0X00000000	0
ALUSrc	0X00000000	0
RegWrite	0X00000001	1
ALUControl	0X00000007	7
Zero	0X00000001	1
branchANDzero	0X00000000	0

Signals

Figura 14 – Sub-Janela de Sinais (Signals)

Sub-janela do Diagrama da arquitectura

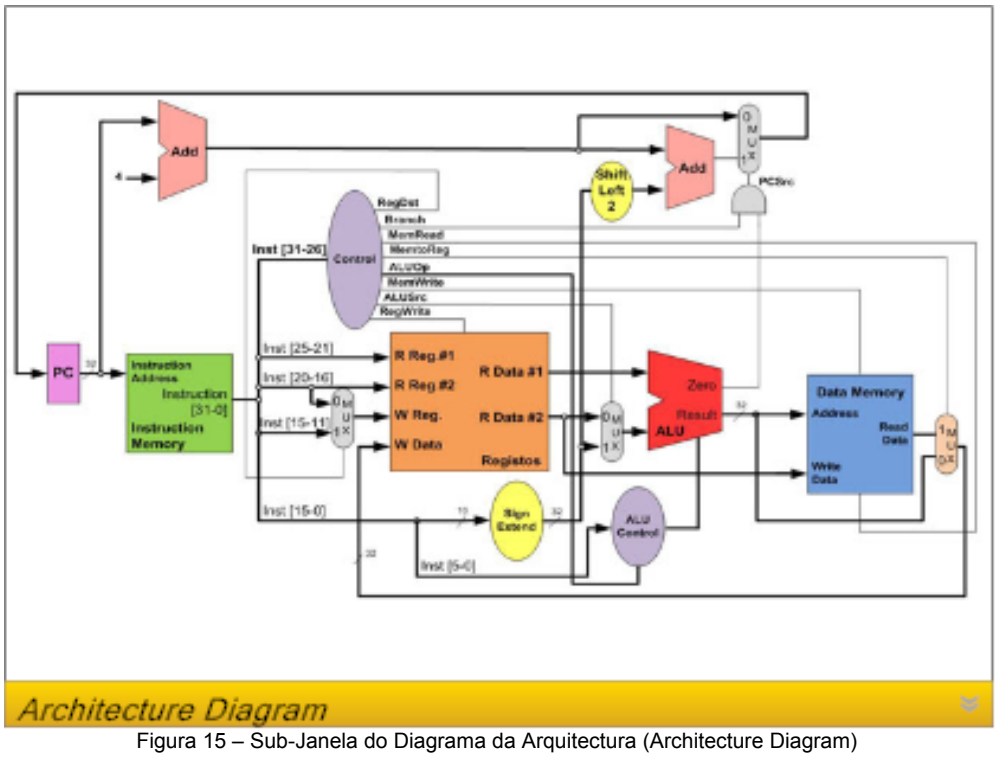


Figura 15 – Sub-Janela do Diagrama da Arquitectura (Architecture Diagram)

A sub-janela do diagrama da arquitectura é uma janela opcional apenas com o intuito de possibilitar a integração de uma imagem que o utilizador tenha e que represente o diagrama de blocos da interligação dos componentes de um processador para assim mais facilmente possa analisar e compreender os resultados obtidos dos sinais lidos e apresentados nas tabelas das restantes sub-janelas.

Na figura 15 poderá visualizar o diagrama da arquitectura MIPS Single Cycle, aonde pode observar os deversificados blocos que constituem esta arquitectura (PC, Memórias, Registos, ALU, Control Unit, etc…).

Para carregar e visualizar uma imagem representativa da arquitectura do processador nesta janela, clique com o rato no interior desta janela e identifique a localização do ficheiro pretendido.

O programa iCmips possui activo por defeito a disposição e redimensionamento das sub-janelas na área das sub-janelas (ver o Janela de Preferências → Interface), no entanto o utilizador pode reordenar a disposição das mesmas, assim como a sua dimensão. Para mover uma sub-janela clique sobre a barra de título da sub-janela com o botão esquerdo do rato e mantenha-o pressionado enquanto arrasta a janela para a posição desejada. Para redimensionar a janela clique com o botão direito do rato sobre a barra de título da sub-janela e mantendo pressionado o botão efectue movimentos horizontais ou verticais com o rato até atingir a dimensão pretendida.

No directório de instalação do iCmips dentro da pasta “Diagrams” o utilizador pode encontrar já o diagrama para um processador MIPS Single Cycle e outro para um processador Multi-thread com 3 pipelines.

Janela de Preferências

Para que o programa iCmips funcione correctamente, é necessário antes tomar uma pequena atenção às configurações do software.

Como é compreensível, podem ser implementadas centenas de variantes de processadores de arquitectura MIPS, sendo apenas a imaginação o limite. Assim para que o software iCmips, não esquecendo que é uma ferramenta para visualização de valores de sinais reais que se encontram na placa DETIUA-S3, é necessário informar a ferramenta de como é que os sinais foram ligados e tipo de componentes foram usados no processador MIPS implementado.

A janela de Preferências do iCmips possui 4 painéis de configuração, sendo 3 deles (User MIPS, Ports Configuration e Instructions) dedicados unicamente à discrição das características da arquitectura do processador implementado e um outro (Interface) para fins de configuração de algumas das opções do software iCmips.

O painel de configuração da arquitectura do processador (User MIPS)

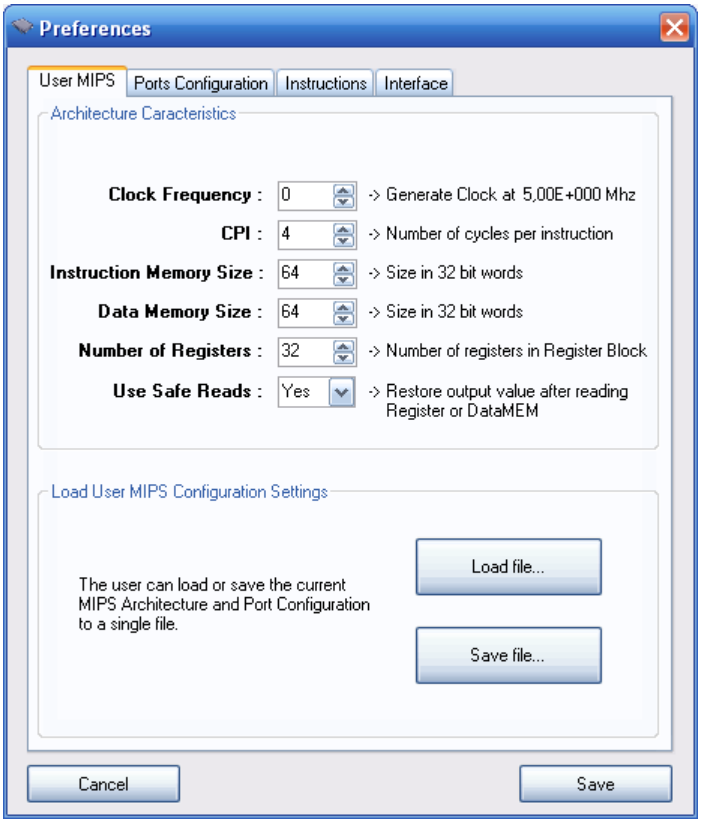


Figura 16 – Painel de configuração da arquitectura MIPS da Janela de Preferências

A figura 16 demonstra o painel de configuração da arquitectura do processador implementado. É neste painel que o utilizador vai declarar as características do seu processador indicando a frequência base de relógio que pretende aplicar no processador (Clock Frequency), o número de ciclos necessários para processar uma instrução (CPI), a dimensão da memória de instruções (Instruction Memory Size) e memória de dados (Data Memory Size), o número de registos que o processador possui (Number of Registers) e se deve efectuar leituras seguras do processador (Use Safe Reads).

A definição de leituras seguras (Use Safe Reads) é uma propriedade implementada que em conjunção com os módulos "SetRegister.vhd" e "SetDataMEM.vhd", possibilita ao utilizador que após a leitura dos registos ou da memória de dados, o valor à saída destes blocos seja reposto, ou seja, para efectuar a leitura dos registos ou da memória de dados são gerados múltiplos ciclos de relógio e colocado à entrada destes módulos por cada ciclo o valor do registo ou endereço de memória pretendido ler, o que faz com que após a última leitura os valores à saída destes blocos possam não ser os mesmos que os valores iniciais corrompendo o processamento da instrução.

Neste painel encontra ainda a possibilidade de guardar (ou ler) um ficheiro de configuração automática das características do processador, de instruções e das ligações efectuadas às portas. Para fazer uso desta funcionalidade o utilizador só tem de clicar sobre um dos botões e seleccionar um dos ficheiros de configuração do iCmips que possuem uma extensão ".ini".

Na pasta "User Settings" que pode ser encontrada no directório de instalação do iCmips, existem 2 ficheiros de configuração: o "MIPS32_Single_Cycle_Config.ini" deve ser usado com um dos processadores MIPS Single Cycle que podem ser implementados usando as bitstreams distribuídas conjuntamente com o iCmips (veja o conteúdo da pasta "Bitstreams" que se encontra no directório de instalação do iCmips) e "MIPS32_Multi-thread_3_Stage_Pipelined.ini" para a bitstream "MIPS32_Multi-thread_3_Stage_Pipelined.bit".

O painel de configuração das portas (Ports Configuration)

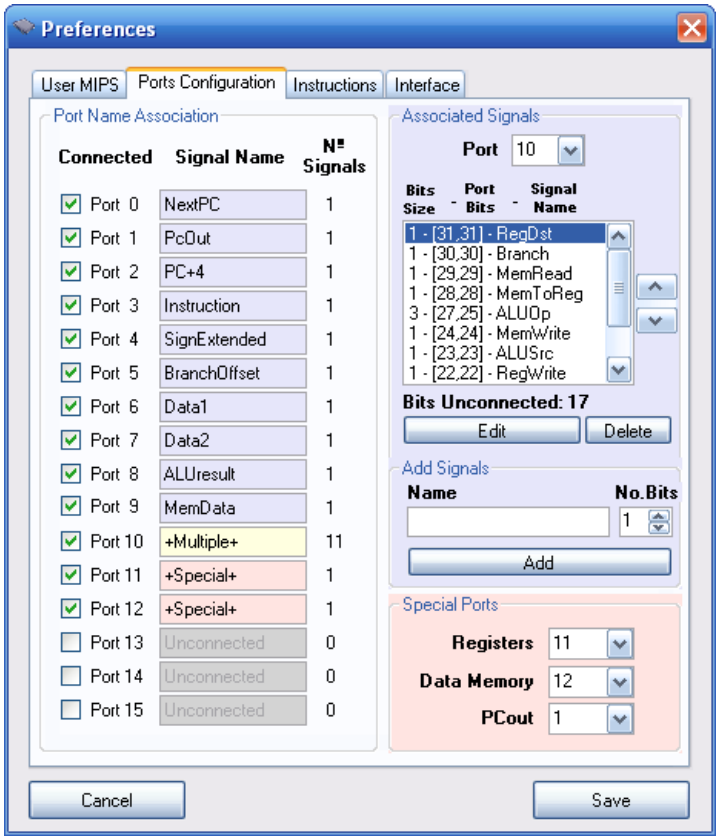


Figura 17 – Painel de configuração das portas de comunicação da Janela de Preferências

É no painel de configuração das portas que o utilizador descreve como efectuou as ligações no seu projecto de modo a que o programa iCmips 1.0 seja capaz de interpretar correctamente o processamento dos valores dos sinais lidos.

À esquerda na figura 17, o utilizador deve declarar quais portas é que se encontram ligadas e qual o nome do sinal associado. Existem 3 possibilidades de configuração de uma porta:

1. Pode ser uma porta com um único sinal de 32 bit;
2. Pode ser uma porta dedicada ("+Special+") à leitura de um dos componentes (Registos ou Memória de dados);
3. Pode ser uma porta com múltiplos sinais associados ("+Multiple+").

Quando activa uma porta, por defeito, a porta fica com o nome predefinido "Port n" (sendo "n" o numero da porta) e com apenas um sinal de 32 bit associado. Para adicionar ou editar os sinais associados à porta activada clique no botão editar na área de "Associated Signals", modifique a dimensão de bits para o sinal e seu nome. Para guardar os novos parâmetros carregue no botão "Save". Se redefiniu a dimensão de bits do sinal activado, então significa que passam a existir ainda bits disponíveis nessa porta para associar

novos sinais, o utilizador pode agora adicionar todos os outros sinais que pretende. Verifique por exemplo a situação da porta 10 na figura 17. Mesmo depois de adicionados os sinais associados a uma das portas podem ser reordenados bastando para isso seleccionar o sinal que pretende reordenar e fazer uso dos botões cima/baixo que se encontram ao lado da caixa de listagem de sinais associados.

O utilizador encontra ainda neste painel outra área de configuração denominada de portas especiais "Especial Ports". Nesta área o utilizador pode identificar, caso faça uso, quais as portas que foram atribuídas a leitura de Registos, Memória de dados e qual das portas corresponde à saída do program counter ("PCout"). O utilizador ao efectuar estas escolhas irá informar o programa iCmips de quais portas irá receber dados para o correcto funcionamento do algoritmo de processamento de dados e visualização.

O utilizador deve possuir muita atenção relativamente à configuração dos sinais associados a cada porta. A má configuração levará à visualização de dados falsos nas tabelas das sub-janelas de dados.

Exemplo de configuração de portas:

Na figura seguinte (Figura 18) é-lhe apresentado parcialmente o código VHDL utilizado no projecto "MIPS32_Single_Cycle_Basic" que deu origem à configuração de portas que é visível na Figura 1 do anexo 4.

```
-- Output signals to interface --
    Port_0 <= NextPC;
    Port_1 <= PCOut;
    Port_2 <= PC4;
    Port_3 <= Instruction;
    Port_4 <= SignExtended;
    Port_5 <= BranchOffset;
    Port_6 <= ALUData1;
    Port_7 <= Data2;
    Port_8 <= ALUResult;
    Port_9 <= MemData;
    Port_10 <= RegDst & Branch & MemRead & MemtoReg & ALUOp
                    & MemWrite & ALUSrc & RegWrite & -- Control Unit
               ALUCtrlCode &   -- ALU Control Code
               Zero &          -- ALU
               BranchAndZero & -- AND
               '0' & X"0000" ; -- BITS NAO ATRIBUIDOS
    Port_11 <= RegisterData;
    Port_12 <= DataMemoryData;
```

Figura 18 – Codigo VHDL de como são ligados os sinais do processador às portas de leitura de sinais do protocolo de comunicação.

O painel de configuração de instruções (Instructions)

O painel de configuração de instruções dá ao utilizador a possibilidade de (re)definir as instruções que o seu processador consegue interpretar.

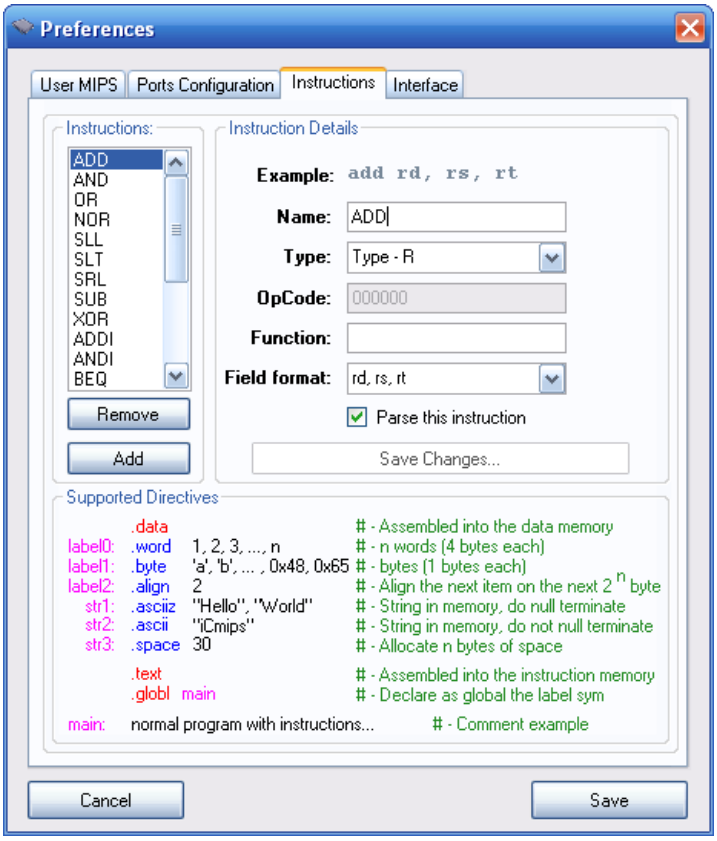


Figura 19 – Painel de configuração das instruções da Janela de Preferências

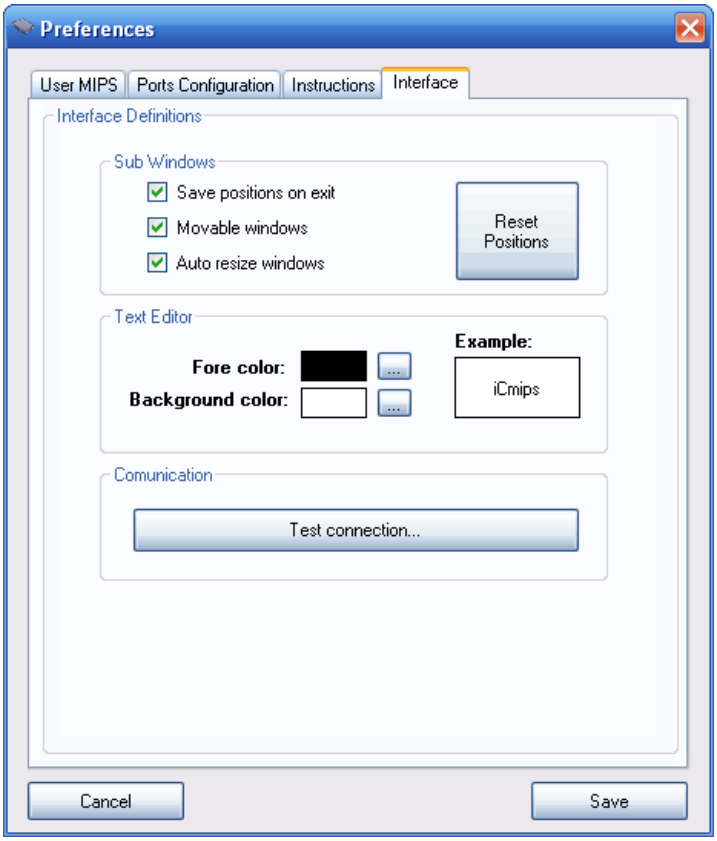
Por omissão o software iCmips apenas permite configurar instruções MIPS de 32bit, do tipo R, I ou J (dropdown menu "Type" e "Field Format" na figura 19) possuindo algumas das instruções aritméticas, lógicas e acesso à memória adicionadas. Para editar alguma das instruções já definidas, seleccione a instrução pretendida da lista que se encontra na área "Instructions" e modifique os parâmetros na área "Instruction Details". Não se esqueça de activar a verificação de sintaxe para a instrução caso queira que o programa seja capaz de analisá-la e transformá-la em código máquina (se não o fizer, obterá um erro de parsing no editor de texto).

Para adicionar novas instruções comece por clicar no botão "Add" e de seguida introduza/seleccione os parâmetros correctos na área "Instruction Details".

Ainda neste painel, encontra um pequeno texto ilustrativo das directivas suportadas pelo interpretador de sintaxe assembly.

O painel de configuração do iCmips (Interface)

O painel de configuração do iCmips dá possibilidade ao utilizador configurar algumas das funcionalidades de acessibilidade embutidas no programa.



Figuras 20 – Painel de configuração da interface iCmips da Janela de Preferências

O conteúdo da área "Sub Windows" permite ao utilizador efectuar escolhas sobre se pretende guardar a posição das sub-janelas à saída do programa, se pretende permitir o movimento e redimensionamento das sub-janelas ou voltar a repor a dimensão e posição por defeito das sub-janelas (ver checkboxs da figura 20 de cima para baixo e botão "Reset Positions").

Na área "Text Editor" pode escolher a cor para o fundo e texto do editor de texto.

Encontra ainda neste painel um botão que lhe permite efectuar um teste de comunicação com a placa DETIUA-S3. Preccione este botão e aguarde no máximo 5 segundos para receber informação sobre o estado de conectividade com a placa, da comunicação usb e se o protocolo implementado na placa é o correcto.

Neste momento encontra-se pronto para começar a trabalhar e descobrir as capacidades do software iCmips. Para isso comece por carregar para a placa DETIUA-S3, um dos processadores fornecidos e carregue os ficheiros assembly, de configuração e de diagrama de arquitectura associados para o mesmo processador.

Após o carregamento de todos os ficheiros:

- Efectue o parser do código assembly e verifique os códigos máquina das instruções e os valores da memória de dados.
- Programe as memórias do processador, transferindo os valores gerados.
- Faça um reset ao processador antes de iniciar a gerar ciclos de relógio.

Configuração e Uso do Hardware

A placa DETIUA-S3

Instalação dos drivers

Após a instalação do software iCmips deve efectuar a ligação da placa DETIUA-S3 com o computador através do cabo USB (tipo A-B). Após ligar a placa de prototipagem, por omissão o sistema operativo vai detectar um novo dispositivo de hardware e dará inicio ao processo de instalação de drivers para esse específico hardware, neste caso a placa DETIUA-S3 (Figura 21).

Figura 21 – PopUp de alerta do sistema informando que foi detectada a placa DETIUA-S3

Passo 1

Figura 22 – Janela do assistente de instalação do novo hardware encontrado

Clique no Popup informativo (Figura 21) que surgiu na barra de tarefas do Windows para fazer surgir a janela do assistente de instalação de novo hardware encontrado (Figura 22).

No assistente é apresentada a questão:

"O Windows pode ligar Windows Update para procurar software?"

Seleccione a opção *"Não, não desta vez"* e clique no botão seguinte para continuar a instalação.

Passo 2

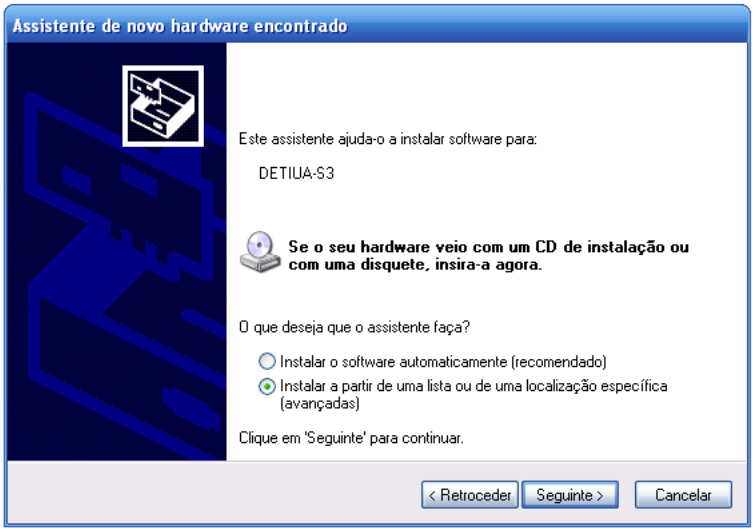


Figura 23 – Janela de opções de instalação do assistente de instalação de novo hardware

Na nova janela que surge, o Windows procura saber qual método que o utilizador pretende para efectuar a instalação dos drivers do hardware (Figura 23).

Seleccione a segunda opção:

"Instalar a partir de uma lista ou de uma localização específica (avançadas) ".

Clique em *"Seguinte >" para proceder ao próximo passo*.

Passo 3

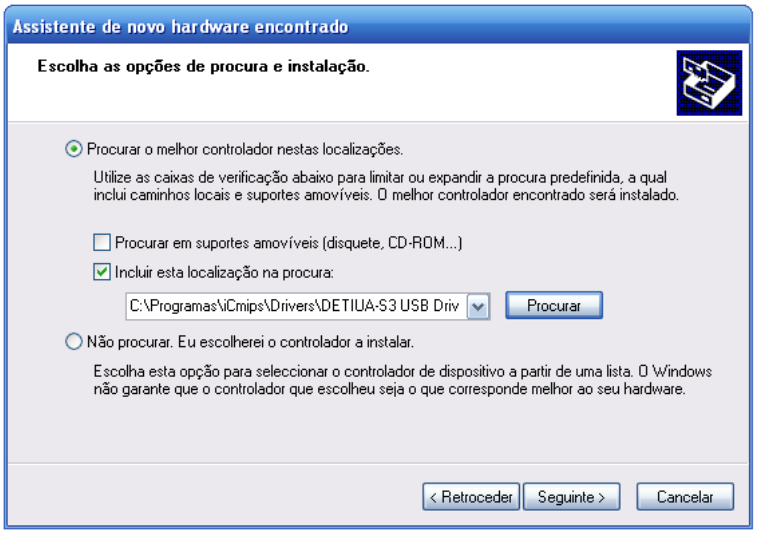


Figura 24 – Janela de escolha da localização dos drivers para instalação do assistente

Neste passo (ver Figura 24), vai seleccionar no assistente que pretende *"Procurar o melhor controlador nestas localizações" e apenas activar a opção "Incluir esta localização na procura:"*. Clique no botão *"Procurar"* para indicar a pasta que contém os drivers para a placa.

Os drivers para instalação e configuração da placa DETIUA-S3 podem ser encontrados no directório de instalação do iCmips (por exemplo: "c:\programas\iCmips\Drivers\DETIUA-S3 USB Driver") ou online [4].

Concluídas as escolhas pressione *"Seguinte >"* para efectuar a instalação.

Passo 4

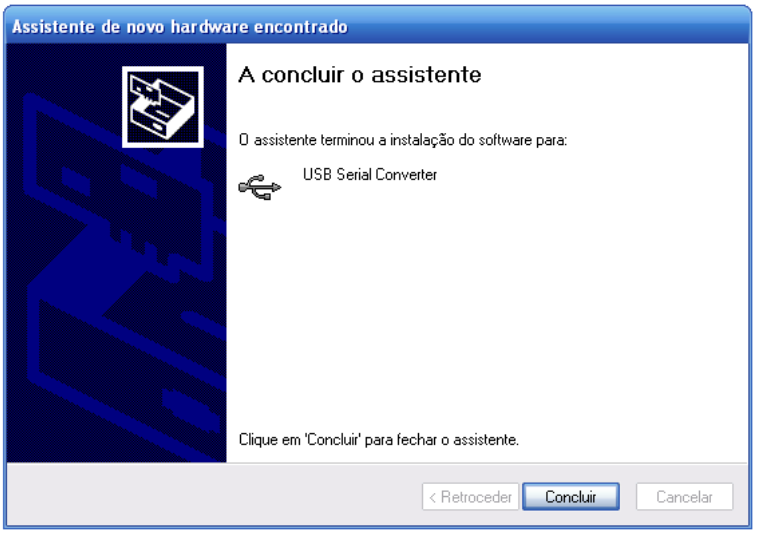


Figura 25 – Janela final de conclusão da instalação do novo hardware encontrado

Visualizará uma janela informativa do progresso da instalação antes de surgir a de conclusão do assistente (Figura 25).

O sistema operativo vai disparar um novo aviso de *"Novo hardware encontrado"* (Figura 26) mas desta vez relativo a *"USB Serial Port"* que corresponde à necessidade de instalação do driver de controlo de comunicações USB serial.

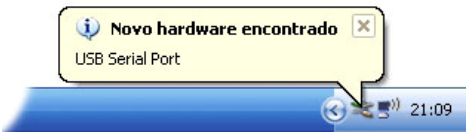


Figura 26 – PopUp de alerta do sistema informando que foi detectado um dispositivo USB Serial Port

Efectue novamente de igual forma todos os passos seguidos até agora (Passos 1 a 3) para efectuar a instalação deste dispositivo.

Concluída a instalação dos drivers no sistema operativo, deverá ser capaz de usar o software ***iCmips*** através do menu ***Tools→Test Comunication*** e verificar a conectividade com a placa.

Como usar a placa

Nesta secção irá encontrar informação de como funcionar com a placa DETIUA-S3, não sendo objectivo descrever todas as capacidades da placa mas apenas o que é necessário saber para o desenvolvimento de um processador de arquitectura MIPS fazendo uso do projecto base fornecido e da ferramenta iCmips 1.0. Para mais informações sobre a placa e suas características, consulte os documentos Manual e Tutorial de configuração da placa DETIUA-S3 [1].

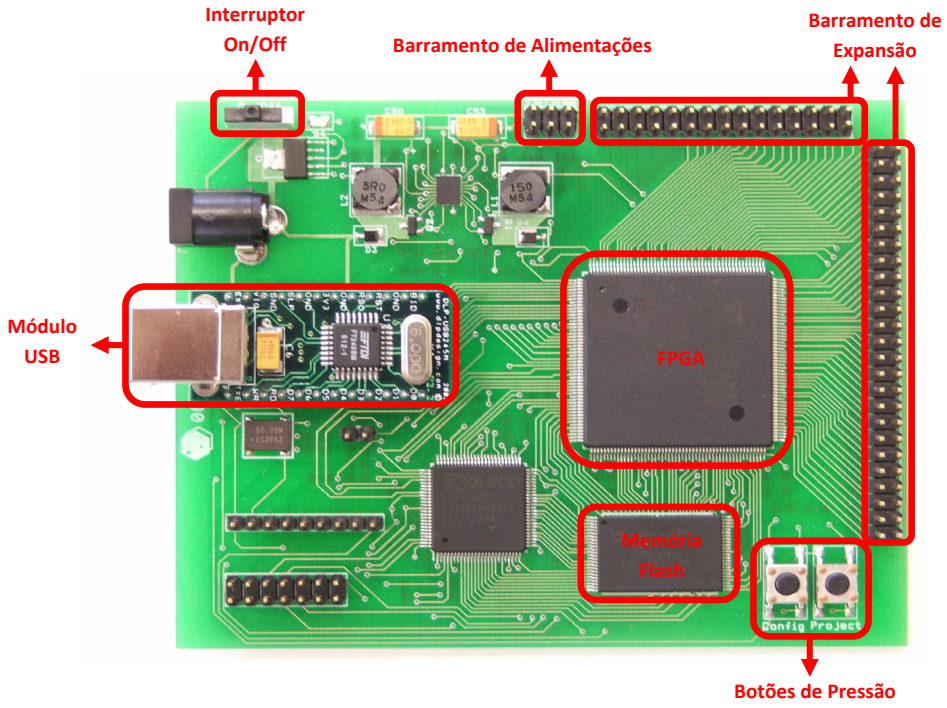


Figura 27 – A placa DETIUA-S3 e identificação de alguns dos seus componentes

Preparar o hardware…

Para começar a usar a placa de prototipagem ligue-a ao computador fazendo uso do cabo USB para interligar a porta USB do computador e o módulo USB da placa. Se for a primeira vez que o faz, consulte neste documento o capítulo "A placa DETIUA-S3 -> Instalação dos drivers".

Opcionalmente pode adicionar, através do barramento de expansão, a mini-placa de expansão de visualização do estado de funcionamento da placa (Figura 28). A mini-placa é constituída unicamente três por LEDs luminosos que dão a conhecer ao utilizador o estado de processamento da placa DETIUA-S3. O significado dos leds activos pode ser encontrado na tabela 4.

Figura 28 – Mini-Placa de expansão de LEDs para sinalização do estado de funcionamento da placa DETIUA-S3. Em cima vista frontal, a meio vista lateral e em baixo vista inferior da placa.

Estado	*LEDs Activos*	*Significado*
1-●○○	Verde	A placa encontra-se pronta a receber e efectuar operações.
2-○●●	Amarelo e Vermelho	A placa encontra-se à espera de receber informação necessária relativa a uma operação.
3-○○●	Vermelho	A placa encontra-se em funcionamento executando uma operação pedida.

Tabela 4 – Tabela de descrição dos estados de funcionamento da placa DETIUA-S3 segundo os LED activos

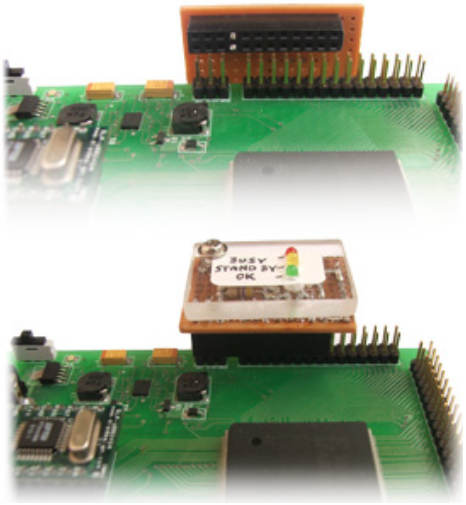


Figura 29 – Demonstração de como a mini-placa de expansão encaixa na placa DETIUA-S3. Em cima a posição de encaixe da mini-placa no barramento de expansão, em baixo a mini-placa ligada à placa DETIUA-S3.

A mini-placa encaixa no barramento de alimentações e parcialmente sobre o barramento de expansão superior. Para ligar a placa de expansão à DETIUA-S3 **deve possuir a placa desligada!** Visualize na figura 29 como ligar a mini-placa.

Trabalhando com a placa…

A placa DETIUA-S3 encontra-se equipada com uma memória flash (ver Figura 27) na qual são armazenadas bitstreams de configuração da FPGA. Por omissão existe um conjunto inicial de sectores reservados na memória que contém uma bitstream que é designada a bitstream por omissão.

A bitstream por omissão possui implementado um protocolo específico de comunicação que permite a comunicação com o computador para fins de transferência de dados, ou seja, o utilizador pode enviar e receber dados e armazenar novas bitstreams [1].

O projecto base disponibilizado implementa uma versão melhorada do protocolo de comunicação, com a capacidade de comunicação com um processador de arquitectura MIPS (ver figura 30). O protocolo de comunicação comporta-se como um intermediário entre o computador, memória flash da placa DETIUA-S3 e o processador implementado.

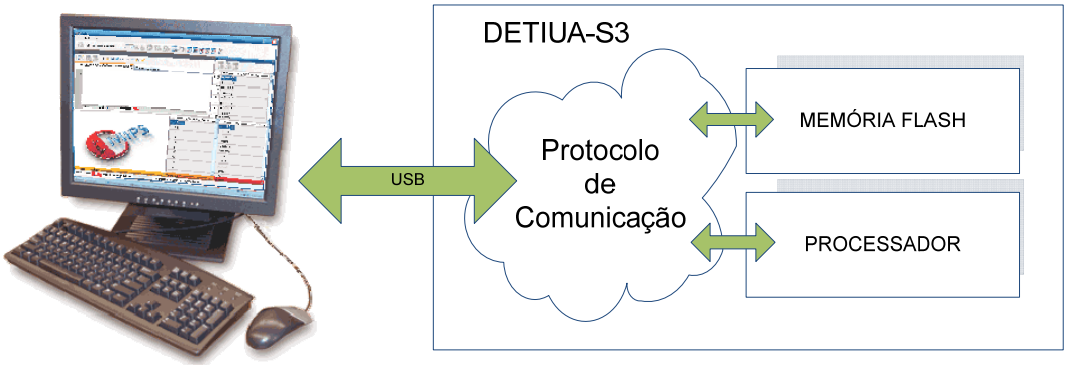


Figura 30 – Diagrama de comunicações do protocolo implementado

O processo de comunicação inicia-se quando é enviada uma mensagem pelo computador para a placa DETIUA-S3. O protocolo de comunicação, implementado juntamente com o processador na FPGA, recebe a mensagem contendo uma operação, descodifica-a e executa-a.

Existem diversas operações permitidas (ver tabela em anexo 2). A título de exemplo, numa operação com o processador de leitura de dados (operação MIPS READ DATAMEMORY) o protocolo gera sinais para o processador podendo ler dados do mesmo e guardar na memória flash (ou ao contrário, ler da memória flash e transmitir para as memórias do processador). Posteriormente, noutra operação com a placa (operação READ FLASH) o conteúdo da memória flash é lido para o computador, podendo nessa altura descodificar e analisar os dados com a aplicação iCmips. Observe no anexo 3 o diagrama de estados do protocolo de comunicação para melhor compreensão da sequência de operações do protocolo implementado.

O utilizador após o desenvolvimento do processador de arquitectura MIPS em VHDL, integra-o no projecto base, sintetiza-o com as ferramentas adequadas e pode transmitir a bitstream gerada através do iCmips 1.0 ou da ferramenta PBM para a placa DETIUA-S3 e esta ficará armazenado na memória flash numa secção dedicada às bitstreams do utilizador.

Para que a FPGA fique devidamente configurada a bitstream do utilizador ou a bitstream por omissão deve ser carregada. Para isso, o utilizador possui dois botões de pressão que disparam um destes procedimentos. O botão denominado de **“Config”** efectua o carregamento da bistream por defeito e o botão **“Project”** a bitstream do utilizador.

De seguida encontra a descrição dos passos para efectuar a transmissão e execução do seu projecto na placa DETIUA-S3:

1. Quando liga a placa DETIUA-S3 no interruptor on/off, inicialmente deve carregar no botão **"Config"** para poder inicializar o protocolo de comunicação e transmitir para a placa o seu projecto.
2. Utilize as ferramentas iCmips (Menu "Tools"→"Transfer Bitstream…") ou o PBM (Menu "Bitstreams"→"Upload user bitstream") para transmitir o seu projecto para a placa.
3. Concluída a transferência de dados, pressione o botão **"Project"** para efectuar a configuração da FPGA com o seu projecto e poder usar a ferramenta iCmips 1.0 de modo a visualizar os valores de sinais e memórias do processador desenvolvido.

Sempre que transmitir uma nova bitstream não se esqueça de **activar antes a bitstream por defeito**, pressionando o botão "Config".

Projecto base

Neste subcapítulo poderá encontrar informações relativas ao mecanismo implementado para efectuar as leituras e escritas de dados do processador assim como fazer uso do projecto base em VHDL para introduzir um novo processador para execução, testes e análise com a ferramenta iCmips 1.0.

Localização ficheiros:

O projecto base foi desenvolvido na ferramenta Xilinx ISE 9.2, sendo por isso encontrados ficheiros próprios deste software no meio dos ficheiros VHDL.

O utilizador pode encontrar na pasta de instalação do iCmips o directório "VHDL" dentro do qual encontrará outro: "BASE_PROJECT" ("%Programas%\iCmips\VHDL\BASE_PROJECT") que contém os ficheiros base (VHDL e ISE Project) para criação de um novo projecto (ver Figura 31).

***ATENÇÃO**: Antes de alterar os ficheiros fornecidos, é aconselhável que efectue uma cópia para outra localização e trabalhe sobre a cópia efectuada. Para usar o Xilinx ISE não pode copiar para um destino em que no seu caminho completo possua espaços, pois irá obter erros da ferramenta.*

*"C:\projectosISE\o meu projecto\BASE_PROJECT" >> **ERRO***
*"C:\projectosISE\o_meu_projecto\BASE_PROJECT" >> **OK***

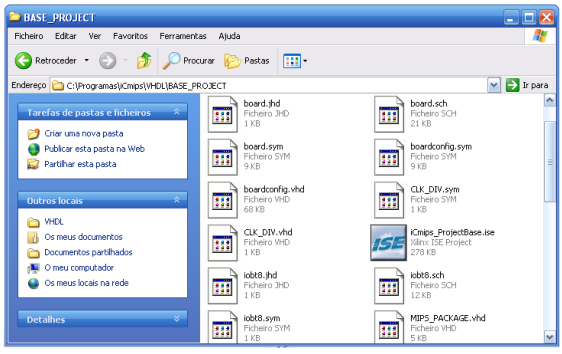


Figura 31 – Explorador de ficheiros do Microsoft Windows XP visualizando alguns dos ficheiros do Projecto Base

Criação de um novo projecto usando Xilinx ISE

Se possuir a ferramenta Xilinx ISE 9.2 ou superior instalada, pode abrir o ficheiro "***iCmips_ProjectBase.ise***" e terá o projecto pronto a utilizar podendo saltar os próximos passos.

Passo 1

Abra o Xilinx ISE e faça um novo *"File"→"New Project"* e configure o menu que lhe aparece conforme as suas necessidades (ver Figuras 32 - A).

Na escolha de opção: "*Top-Level Source Type"* seleccione *"Schematic" e prima "Next >" para continuar.*

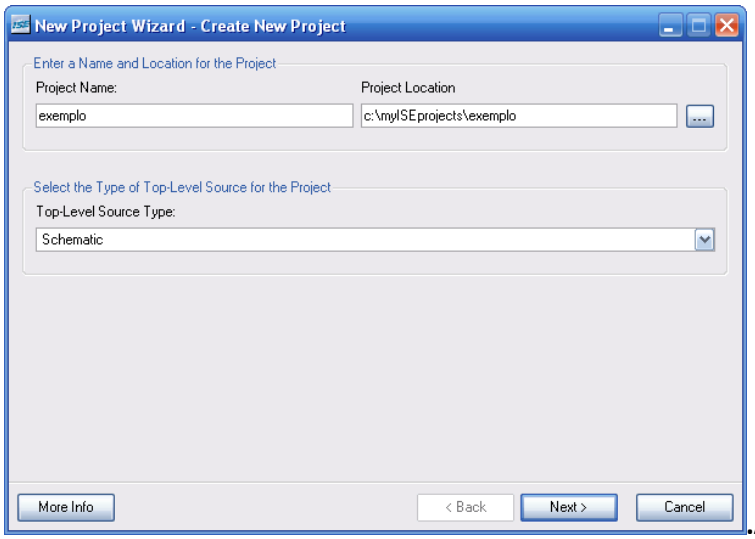
Passo 2

Na nova janela que aparece, vai configurar o projecto para ser sintetizado para as características da placa DETIUA-S3 (Figuras 32 – B). Para isso efectue as seguintes escolhas:

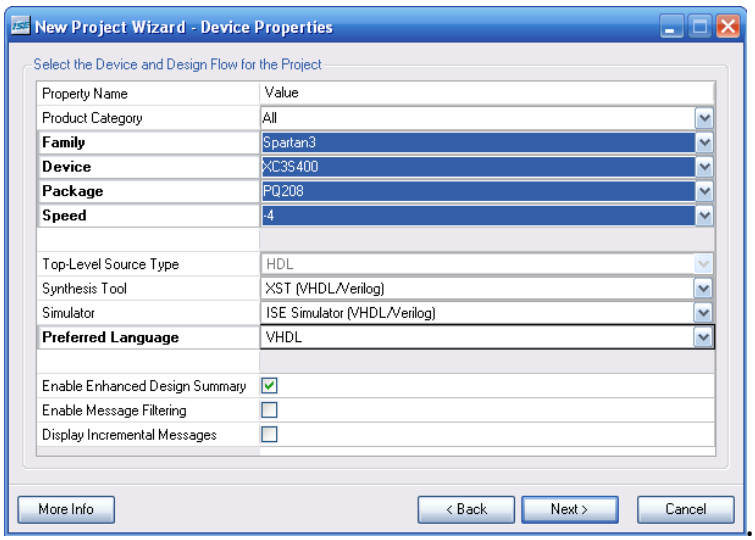
Family = Spartan3
Device = XC3S400
Package = PQ208
Speed = -4
Preferred Language = VHDL

Passo 3

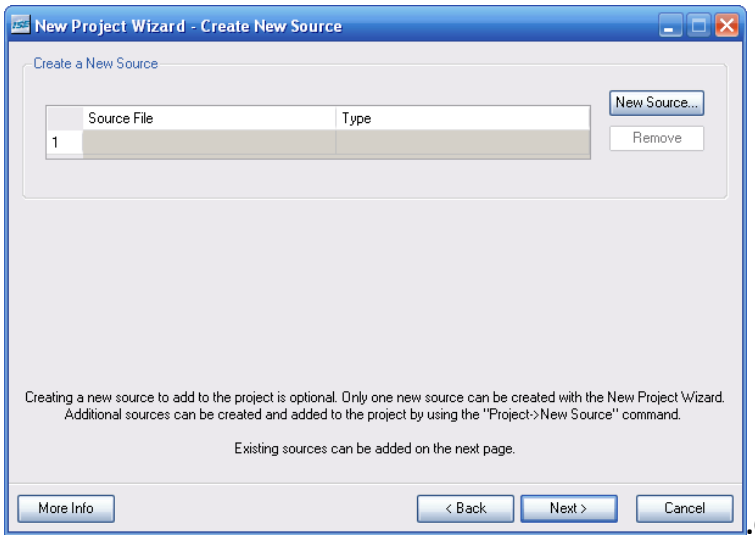
Neste momento possui a possibilidade de criar no projecto novas fontes de código VHDL (Figuras 32 – C), mas como se pretende adicionar as já existentes, faça *"Next >" para o próximo passo.*



.A



.B



.C

Figuras 32 – Janelas do assistente de novo projecto do Xilinx ISE:
A- Criar novo projecto;
B- Propriedades da placa a usar;
C- Adição de novos códigos fonte.

Passo 4

Neste passo o utilizador vai clicar no botão *"Add Source"* e vai à pasta *"iCmips\VHDL\BASE_PROJECT"* seleccionar todos os ficheiros lá existentes.

(Existe a possibilidade de o professor disponibilizar outras fontes, de um projecto especifico para a disciplina no entanto o processo de adição é o mesmo)

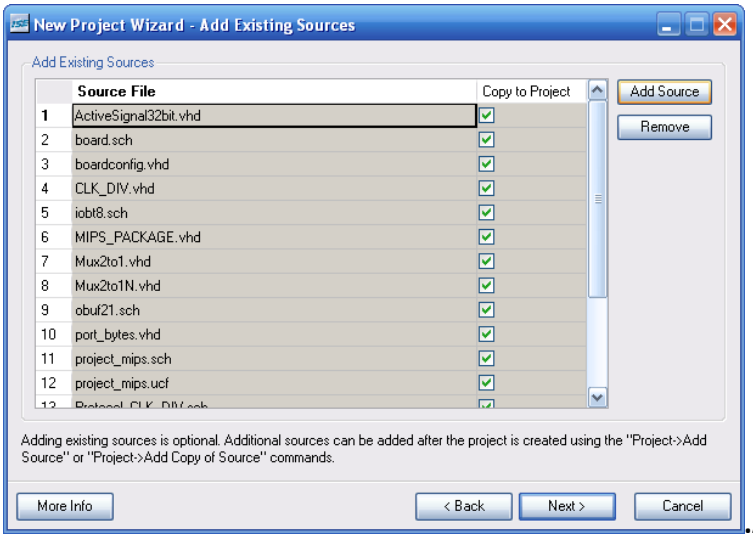
Feito isto, a sua janela terá um aspecto idêntico à Figuras 33-A. Carregue em *"Next>"* para continuar.

Passo 5

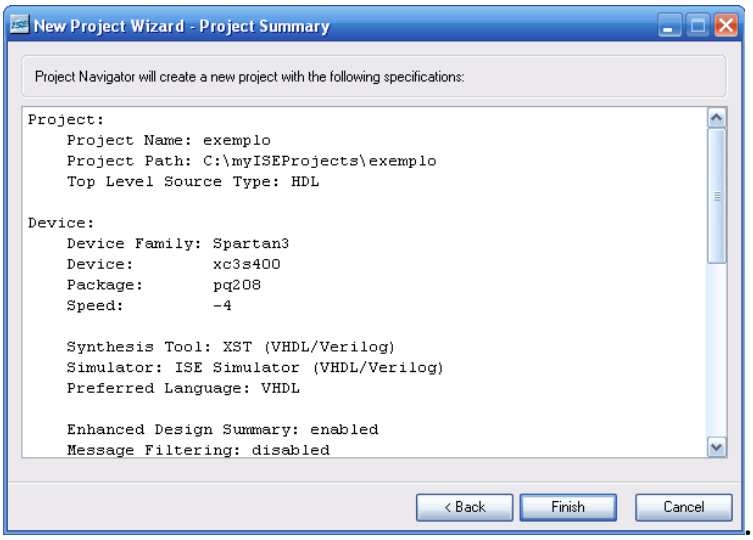
Neste passo apenas obtém um sumário das opções efectuadas até ao momento (Figuras 33-B). Clique *"Finish"* para concluir e dar inicio à importação do código fonte do projecto base.

Passo 6

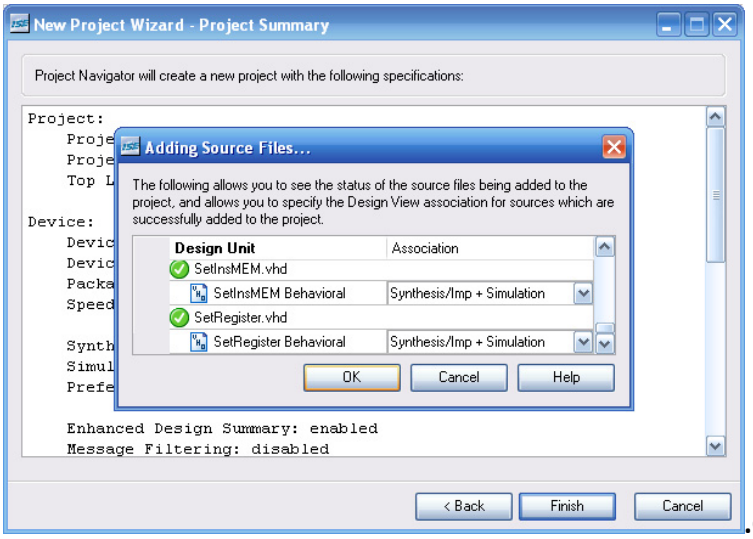
O Xilinx ISE vai analizar todos os códigos fonte escolhidos, e processa-los (Figuras 33-C). No fim aparecerá uma janela em que apenas terá de clicar no botão *"OK"* para concluir a criação de um novo projecto.



.A



.B



.C

Figuras 33 – Janelas do assistente de novo projecto do Xilinx ISE:
A- Adicionar fontes de código existente;
B- Sumário do projecto;
C- Novas fontes de código adicionadas.

Como usar o Xilinx ISE

Tendo o projecto aberto/criado a ferramenta Xilinx ISE fica com um aspecto semelhante à figura 34.

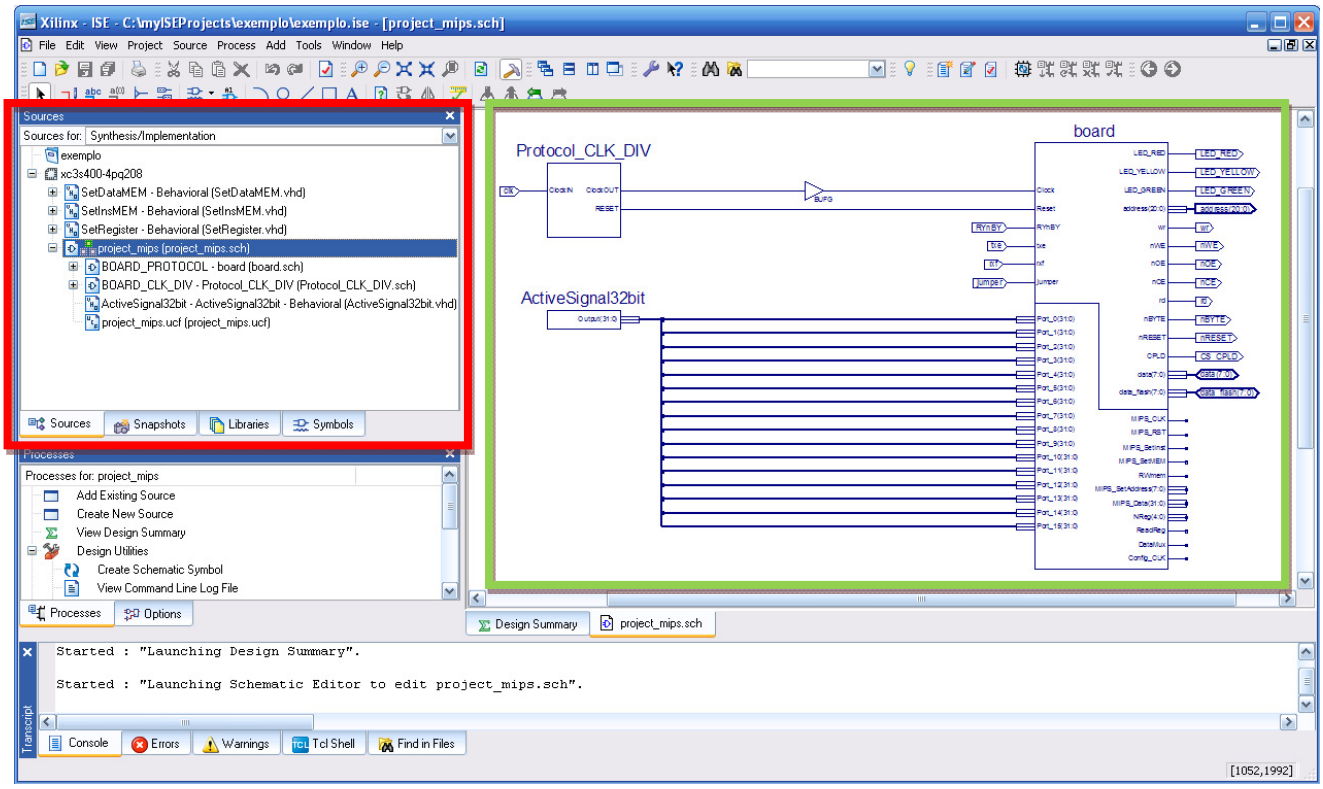


Figura 34 – Ambiente de trabalho da ferramenta Xilinx ISE com o projecto base aberto

Na sub-janela das "Sources" (campo a vermelho na Figura 34), o utilizador pode encontrar os módulos que são incluídos no projecto.

A source principal é denominada de **"project_mips (Project_mips_sch)"**, a qual deve ser declarada como sendo o ***"Top Module"*** *(clique com o botão direito do rato sobre a source e escolha "Set as Top Module")*. Esta source corresponde ao diagrama esquemático dos blocos do projecto. Se fizer duplo clique sobre este módulo, o ISE abrirá o esquemático e poderá visualizá-lo (campo a verde na Figura 34).

No projecto base não possui nenhum processador associado. O utilizador deve criar o seu processador, gerar um símbolo esquemático (selecionar a source correspondente ao seu processador e na sub-janela *"Processes"* fazer duplo clique em *"Design Utilities → Create Schematic Symbol"*) e adicioná-lo no esquemático substituindo o módulo "ActiveSignal32Bit".

Descrição dos módulos do projecto

Nesta secção encontrará alguma informação relativa aos módulos existentes no projecto base e como fazer uso dos mesmos.

O utilizador é livre de visualizar o código VHDL de cada um dos módulos, no entanto, é de todo aconselhável que não efectue nenhuma alteração no mesmo de modo a garantir a sua integridade e funcionalidade.

O divisor do sinal de relógio

Figura 35 – Módulo correspondente ao divisor do sinal de relógio usado no protocolo de comunicação

A placa DETIUA-S3 é equipada com um oscilador de sinal capaz de produzir um sinal de relógio de 80MHz. No entanto o projecto tem como característica funcionar a 5MHz e isto deve-se ao facto da necessidade de obter frequências mais baixas de modo a conseguir-se operar com a memória flash da placa.

Este divisor de relógio (ver Figura 35) possui como entrada o sinal de relógio da placa DETIUA-S3, e à saída um sinal de relógio dividido e o sinal de reset.

A interface com a placa

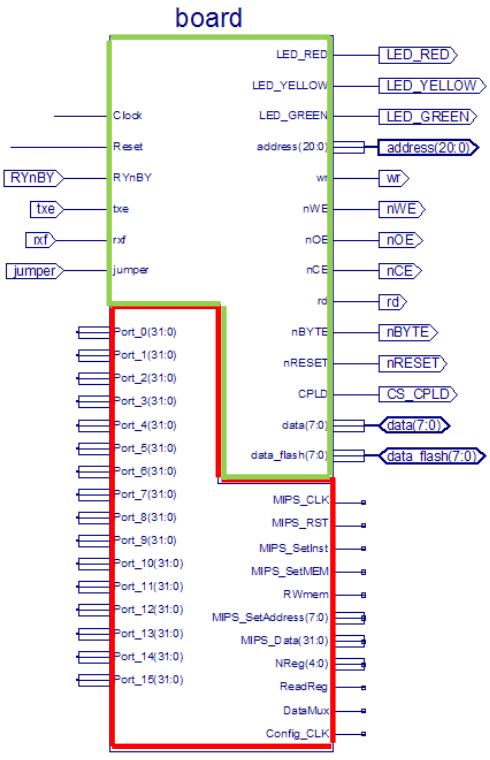


Figura 36 – Módulo de interface entre a placa e o processador

Este módulo é o componente vital para toda a comunicação com a placa DETIUA-S3. O módulo implementa o protocolo de comunicação e encontra-se dividido em duas partes:

- A parte superior (ver Figura 36 a verde) é responsável pela comunicação USB da placa com o computador e a leitura e escrita da memória flash (entre outros não relevantes para o projecto).
- A inferior (ver Figura 36 a vermelho) é a responsável por efectuar a comunicação com o processador implementado.

Na parte superior o utilizador não necessita de alterar nada, mas a parte inferior é a parte relevante ao utilizador. Nela pode encontrar as portas de entrada e saída dos sinais necessários. Do lado esquerdo do módulo utilizador pode ligar todos os sinais que pretende ler do seu processador, tendo para esse efeito 16 portas de 32 bit cada. Do lado oposto, lado direito, encontram-se todos os sinais necessários para controlo do processador. O utilizador pode consultar a tabela 5 para conhecer o significado e efeitos de cada uma destas portas.

Nome da porta	Nº Bits	Significado	Efeito
Port_[0-15]	32	Portas de entrada de 32bit para recepção de sinais oriundos do processador	Quando ligadas, permite visualizar o valor dos sinais associados nas tabelas da ferramenta iCmips.
MIPS_CLK	1	Porta de saída do sinal de relógio que vai alimentar o processador	Produz um ou mais ciclos completos de sinal de relógio, quando no programa iCmips é efectuado o comando de gerar ciclos de relógio.
MIPS_RST	1	Porta de saída do sinal de reset que vai alimentar o processador	Numa operação de reset com o programa iCmips, este sinal é activo e efectuado um ou mais períodos completos de sinal de relógio.
MIPS_SetInst	1	Sinal de controlo para carregamento de instruções na Instruction Memory de um processador	Para poder-se transferir instruções para a memória de instruções de um processador, este sinal é colocado activo, disparando mecanismos que permitem a escrita na Instruction Memory
MIPS_SetMem	1	Sinal de controlo para carregamento de dados na Data Memory de um processador	Para poder-se transferir dados entre a memória de dados de um processador e o programa iCmips, este sinal é colocado activo, disparando mecanismos que permitem a escrita ou leitura da Data Memory
RWmem	1	Sinal de controlo que identifica se a memória de dados deve ser lida ou escrita.	Quando activo a '1', a memória de dados é escrita com valores oriundos do iCmips. Caso contrário é efectuada a leitura da memória para o programa.
MIPS_SetAdress	32	Sinal no qual o programa iCmips endereça as posições das memórias para escrita ou leitura.	É um sinal partilhado pelas duas memórias de um processador (memória de instruções e dados) sendo possível endereçar até 2^{32} endereços de memória (parametrizável).
MIPS_Data	32	Sinal no qual o programa iCmips transmite os dados a serem escritos nas memórias.	É um sinal partilhado pelas duas memórias (instruções e dados) que possibilita a escrita de palavras de 32bits na memória (instruções ou dados).
NReg	5	Sinal no qual o programa iCmips identifica qual registo do processador pretende ler.	Sinal capaz de identificar 2^5 registos de um processador para leitura e posterior visualização dos seus valores nas tabelas do programa iCmips.
ReadReg	1	Sinal de controlo do banco de registos do processador	Quando activo a '1', a pedido de uma leitura dos registos do processador pelo iCmips, o controlo de acesso ao banco de registos é alterado de modo a que seja possível obter o valor de todos os registos para visualização no programa iCmips.
DataMux	1	Sinal de controlo de fluxo de informação que é injectada à entrada das memórias.	Quando activo a '1', os dados a serem escritos nas memórias (instruções ou dados) são provenientes do programa iCmips.
Config_CLK	1	Sinal de relógio usado durante operações de leitura ou escrita de dados com o programa iCmips.	Este sinal de configuração serve para poder gerar um sinal de relógio que permita disparar os mecanismos de escrita ou leitura das memórias sem que seja necessária a alteração do estado do sinal de relógio do processador.

Tabela 5 – Descrição de alguns sinais de controlo e comunicação com o módulo do processador

Algumas informações técnicas e notas relevantes do modo de funcionamento do protocolo implementado:

- O Config_CLK possui uma frequência fixa igual à saída do módulo do divisor de relógio (por defeito é de 5MHz);
- O MIPS_CLK possui um frequência máxima igual à saída do módulo do divisor de relógio (5MHz) sendo possível obter múltiplas frequências mais baixas através do programa iCmips (ver "Preferences→User MIPS→Clock Frequency");
- O programa iCmips, faz uso das operações existentes no protocolo implementado. Esta versão é uma evolução do protocolo já existente, com algumas modificações e optimizações. O diagrama da máquina de estados do protocolo pode ser encontrado nos anexos (ver Anexo-1) juntamente com a Tabela de Anexo-2 que faz um resumo das operações permitidas, os seus campos de parâmetros e quais os seus códigos hexadecimais. Estes códigos podem ser usados directamente no terminal do PBM veja como consultando um pequeno tutorial nos anexos (Anexo-3);
- A decisão de sincronização dos flancos do sinal de relógio deve ser cuidada para que o protocolo implementado funcione, para mais informações consulte os diagramas temporais fornecidos no Anexo-4;

Módulos para integração no processador

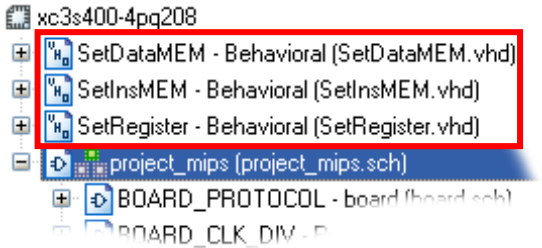


Figura 37 – Dentro do rectângulo vermelho as 3 sources que implementam a interface de controlo da memória de dados, instruções ou banco de registos (respectivamente de cima para baixo)

Foram desenvolvidos 3 módulos (SetDataMEM, SetInstMEM e SetRegister) com o intuito de serem embutidos num qualquer processador de modo a permitirem a acção externa sobre 3 componentes comuns num processador de arquitectura MIPS (Memória de Dados, Memória de Instruções e Banco de Registos, respectivamente). O código fonte desses módulos (Figura 37) pode ser encontrado juntamente com o projecto base para futuro uso e integração na arquitectura do processador do utilizador caso pretendido (exemplo integração na figura 38).

Estes módulos são constituídos por um mecanismo de multiplexers que permutam os sinais das portas de entradas para saída, às quais se devem ligar os sinais respectivos. Para saber qual o papel de cada porta e efectuar as ligações devidas, consulte os comentários inseridos na entidade do código fonte de cada um dos módulos.

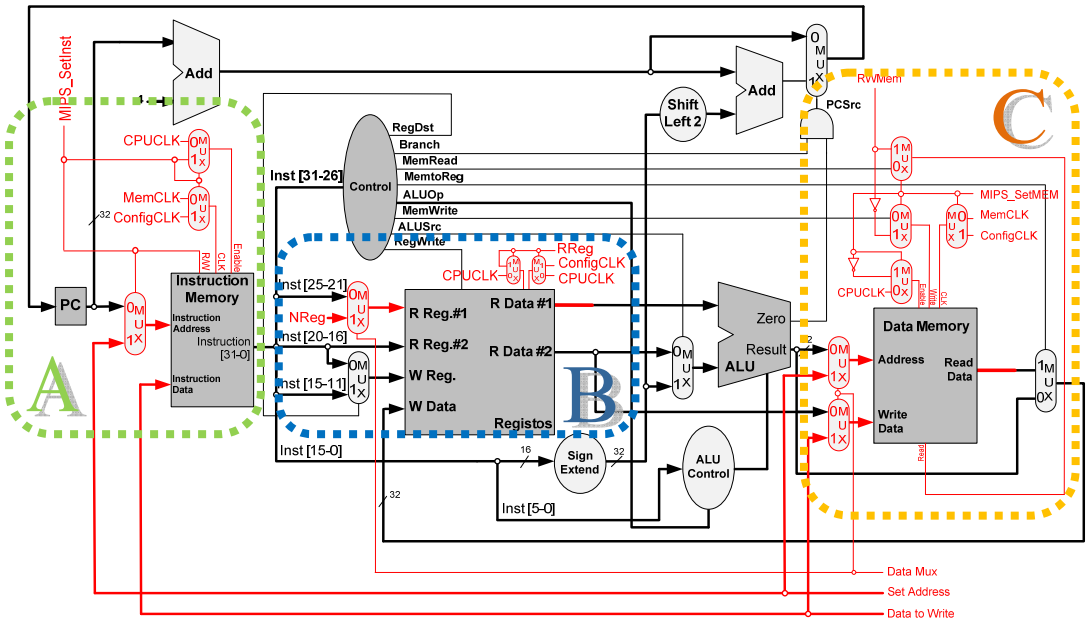


Figura 38 - MIPS Single Cycle Datapath com os módulos SetInstMEM (área A), SetRegister (área B) e SetDataMEM (área C) adicionados

Na figura 38 pode visualizar um exemplo de como podem-se implementar os 3 módulos de controlo (áreas A, B e C) que possibilitam o carregamento de instruções (área A) e dados (área C) e leitura do conteúdo dos registos (área B) e memória (área C).

Os componentes são facilmente adicionáveis em qualquer projecto MIPS (ou outro), sendo introduzidos novos portos de entrada à arquitectura do processador para os sinais: **ConfigCLK**, **MIPS_SetInst**, **ReadReg**, **NReg**, **MIPS_SetMEM**, **RWmem**, **DataMux**, **MIPS_Data** e **MIPS_SetAddress**.

Endereço de Memória (Hex)	Conteúdo da Memória de Dados (Hex)
00	0x04
04	0x10
08	0x00
0C	0x08
10	0x01
14	0x02
18	0x03
1C	0x04
20	0x20
24	0x00

Tabela 6. - Conteúdo da *Data Memory*

Módulo MIPS_PACKAGE

O projecto base possui um ficheiro denominado *"MIPS_PACKAGE.vhd"* que é usado para declarar variáveis e constantes possíveis de usar no projecto, por exemplo a dimensão em bits da arquitectura do projecto e conteúdo das memórias. O conteúdo das memórias (instruções e dados) das bitstreams MIPS32 Single Cycle (Basic e Advanced) fornecidas pode ser visualizado nas tabelas 6 e 7, e consiste num simples programa de leitura e soma de um array de inteiros da memória.

Nº da Instrução	Endereço de Memória (Hex)	Instrução Assembly (valores em hexadecimal)	Instrução (Hex)
0.	00	nop	x00000000
1.	04	lw r1, 0(r0)	x8C010000
2.	08	lw r3, 4(r0)	x8C030004
3.	0C	lw r4, 8(r0)	x8C040008
4.	10	lw r5,20(r0)	x8C050020
5.	14	lw r7,10(r0)	x8C070010
6.	18	Loop: lw r2, 0(r3)	x8C620000
7.	1C	add r4,r4,r2	x00822020
8.	20	add r3,r3,r1	x00611820
9.	24	slt r6,r3,r5	x0065302A
10.	28	beq r6,r7,Loop	x10C7FFFB
11.	2C	sw r4,24(r0)	xAC040024
12.	30	lw r8,24(r0)	x8C080024

Tabela 7. - Conteúdo da *Instruction Memory*

Outros projectos disponibilizados na versão professor

Existem outros projectos disponibilizados juntamente com a instalação da versão professor (*"Teacher"*) do software iCmips. Entre os quais o professor encontra duas versões do MIPS32 Single Cycle (básica e avançada) e um processador MIPS32 Multi-thread com 3 pipelines.

Os ficheiros de cada projecto encontram-se na pasta VHDL na subpasta com o nome associado. Nestes projectos não encontrará os ficheiros associados ao software Xilinx ISE, sendo responsabilidade do professor de criar e importar todas as fontes para um novo projecto *(Não esquecer que o "Project_mips" deverá ser a fonte "Top-Level Module").*

As diferenças entre as duas versões do processador MIPS32 Single Cycle são meramente na quantidade de instruções suportadas (ver tabela 8).

Nome da Instrução	Suportada		
	MIPS32 Single Cycle Basic	MIPS32 Single Cycle Advanced	MIPS32 Multi-thread 3 stage pipelined
LW	X	X	X
SW	X	X	X
BEQ	X	X	X
ADD	X	X	X
SUB	X	X	X
AND	X	X	X
OR	X	X	X
SLT	X	X	X
ADDi		X	
ORi		X	
ANDi		X	
SLTi		X	
LUI		X	
SLL		X	
SRL		X	

Tabela 8 – Instruções suportadas nos processadores incluídos na versão professor.

Os dois processadores MIPS32 Single Cycle implementados já possuem um conjunto de instruções/dados carregados nas suas respectivas *blockRAM* conforme explicado anteriormente.

O processador MIPS32 Multi-thread 3 stage pipelined é uma versão mais complexa e especifica do processador, mais informação sobre a sua arquitectura pode ser encontrada no artigo [6]. Para esta arquitectura não foram usados os módulos de integração no processador que possibilitam a leitura e escrita das memórias (o que nada impede de ser feito) visto que apenas foi disponibilizado para demonstração das funcionalidades do software iCmips.

Bibliografia

Para elaboração deste tutorial foram consultadas as seguintes obras, documentos ou sites online:

[1] Manual e Tutorial de configuração da placa DETIUA-S3:

- http://www.ieeta.pt/~skl/Research/Projects/Manual_Utilizador_pt.pdf (Manual)
- http://www.bernardosilva.net/ (Secção de Projectos → iCmips →Downloads , Tutorial)

[2] Software Prototyping Board Manager 2 (PBM2):

- http://www.ieeta.pt/%7Eskl/CR/DITIUAS3/install%20PBM2.rar

[3] Manual de utilização do Prototyping Board Manager (PBM):

- http://www.ieeta.pt/~skl/Research/Projects/PBM_User_Manual_PT.pdf

[4] Drivers para ligação à DETIUA-S3 via USB:

- http://www.ftdichip.com/FTDrivers.htm
  ou
- http://www.ieeta.pt/~skl/CR/DITIUAS3/d10606.rar
  ou
- http://www.bernardosilva.net/ (Secção de Projectos → iCmips → Downloads)

[5] Microsoft .Net Framework 2.0:

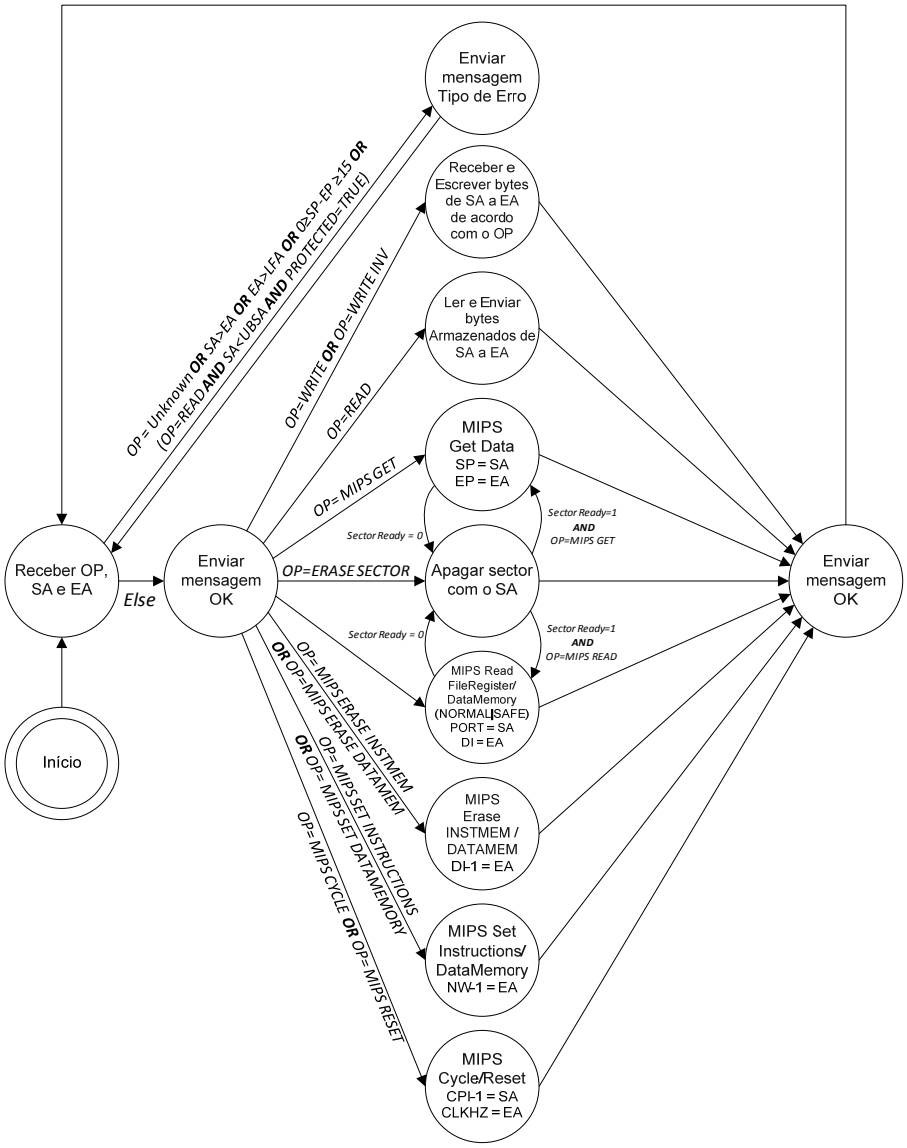
- http://www.microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en

[6] Artigo descrevendo o processador MIPS Multi-thread 3 stage pipelined:

- http://www.bernardosilva.net/ (Secção Projectos → iCmips →Documentos)

Anexos

1 - Diagrama de Estados da FSM do protocolo de comunicação implementado



Legenda:

OP (Operation) - Parâmetro de 1 byte, para especificar a operação;
SA (Starting Address) - Parâmetro de 3 bytes para especificar o endereço da memória flash inicial para a operação solicitada; válido se estiver entre 0 e **EA**;
EA (Ending Address) - Parâmetro de 3 bytes para especificar o endereço da memória flash final para a operação solicitada; válido se estiver entre **SA** e **LFA**;
LFA (Last Flash Address) - Constante específica da placa, que indica o endereço da última posição da memória flash;
UBSA (User Bitstream Starting Address) - Constante específica da placa, que indica o endereço inicial da segunda zona lógica da memória flash;
SP (Start Port) – Parâmetro de 3 bytes para especificar a partir de que porta é para ler e guardar dados na memória flash.
EP (End Port) – Parâmetro de 3 bytes para especificar qual a última porta que se pretende ler e guardar dados na memória flash.
CPI (Cycles per Instruction) – Corresponde ao número de ciclos de relógio que é necessário para executar uma instrução mips.
NW (Number of Words) – Corresponde ao número de words de 32 bits que se pretende processar.
CLKHZ (Clock Hz) – Parâmetro que permite definir o divisor da frequência de relógio que se pretende usar.
DI (Dimension of INSTMEM/DATAMEM) – Parâmetro que permite definir quantas words de 32bit o módulo INSTMEM/DATAMEM consegue armazenar.
Sector Ready – Representa uma flag interna que informa se o sector já foi apagado. Para se escrever num sector é necessário primeiro apaga-lo.

2 - Tabela de operações possíveis com a placa DETIUA-S3

	Operações				
	Comando	**OP (HEX) 1 Byte**	**Campo 1 (HEX) 3 Bytes**	**Campo 2 (HEX) 3Bytes**	**Breve descrição**
**	ERASE SECTOR	04	Start_Addr	End_Addr	• Apaga Sector de memória flash apontado pelo endereço **Start_Addr;** (Start_addr ≥ End_Addr)
	READ FLASH	05	Start_Addr	End_Addr	• Lê da memória flash do endereço **Start_Addr** até ao endereço **End_Addr**; • Número de bytes lidos corresponde a **End_Addr-Start_Addr+1**
**	WRITE FLASH	06	Start_Addr	End_Addr	• Escreve na memória flash do endereço **Start_Addr** até ao endereço **End_Addr**; • Número de bytes a escrever corresponde a **End_Addr-Start_Addr+1**;
**	WRITE INVERSE FLASH	07	Start_Addr	End_Addr	• Idêntico a **WRITE FLASH**, mas escreve os bytes em ordem inversa na memória flash.
*	MIPS CYCLE	08	CPI-1	CLKHZ	• Desactiva o sinal RESET se activo e gera **CPI** ciclos de relógio para uma instrução MIPS • **CLKHZ** possibilita dividir a frequência base do relógio (Freq=$\frac{5}{2^{CLKHz}}\ MHz$, i.e., CLKHZ = 0 dá a frequência base de 5MHz - Período de 200ns) • Devolve Op=01 no final de gerar os ciclos;
*	MIPS RESET	09	CPI-1	CLKHZ	• Idêntico a MIPS CYCLE, só que activa o sinal RESET se este estiver desactivado. • Devolve Op=01 no final de gerar os ciclos;
*	MIPS GET DATA	0A	Start_PORT	End_PORT	• Apaga o sector de memória MIPS_SIGNAL_ADDR = 0x1FC0000 • Grava os valores dos sinais do datapath do MIPS para a memória flash (4 bytes por cada porta * End_Port-Start_Port portas) a partir do endereço MIPS_SIGNAL_ADDR; • Devolve Erro (OP=fe) se 0≤End_Port-Start_Port ≤ 15 • Devolve OP=01 no final da operação
*	MIPS READ FILE REGISTER	0B	Port	DIM-1 (nºRegistos)	• Apaga o sector de memória MIPS_REGS_ADDR = 0x1F80000 • Grava o conteúdo dos Registos do MIPS para a memória flash (4 bytes por cada registo * DIM registos) a partir do endereço MIPS_REGS_ADDR; • Devolve Erro 0xFE se 0≤ Port ≤ 15 • Devolve OP=01 no final da operação
*	MIPS READ FILE REGISTER (SAFE)	1B	Port	DIM-1 (nºRegistos)	• Idêntico a MIPS READ FILE REGISTER, mas gera um CPI no final da leitura para repor os valores à saída do FILEREGISTER
*	MIPS READ DATAMEMORY	0C	Port	DIM-1 (in WORDS)	• Idêntico a MIPS READ FILE REGISTER, mas o endereço de memoria corresponde a MIPS_DMEM_ADDR = 0x1FA0000; • Devolve Erro 0xFE se 0≤ Port ≤ 15 e 0xFD se DIM > DIM(DATAMEMORY definida na package)
*	MIPS READ DATAMEMORY (SAFE)	1C	Port	DIM-1 (in WORDS)	• Idêntico a MIPS READ DATAMEMORY, mas gera um CPI no final da leitura para repor os valores à saída da DATAMEMORY;
*	MIPS SET INSTRUCTIONS	0D	-	DIM-1 (in WORDS)	• Activa o sinal de RESET e carrega ***DIM*** words (cada uma correspondente a uma instrução de 32bit) a partir da posição de memória 0x1E0000 da flash, para o módulo INSTMEM do MIPS. • Para cada Word que é transferida, é gerado ***CPI*** ciclos de relógio à frequência base (5MHz) • Devolve Op=01 no final de todas as operações.
*	MIPS SET DATAMEMORY	0E	-	DIM-1 (in WORDS)	• Idêntico ao MIPS SET INSTRUCTIONS, só que transfere os dados do endereço 0x1F0000 da memória flash para o módulo DATAMEM do MIPS.
*	ERASE INSTMEM	0F	-	DIM-1 (in WORDS)	• Activa o sinal RESET e gera **CPI** ciclos de modo a substituir todas as instruções de INSTMEM (de 0 a **DIM**) por *NOP*'s = 0x00000000. • No final da operação devolve Op=01.
*	ERASE DATAMEM	1F	-	DIM-1 (in WORDS)	• Idêntico a ERASE INSTMEM, mas neste caso elimina todos os dados da DATAMEM substituindo-os por valor zero.
	Mensagens				
**	OP ERROR	00	-	-	• OPCODE desconhecido
	OK	01	-	-	• Operação recebida válida ou completada com sucesso
	JUMPER ERROR	02	-	-	• Jumper não colocado, escrita na primeira zona de memória não permitida sem jumper
	ADDRESS ERROR	03	-	-	• O conjunto de endereços introduzido não é válido
*	NUMBER OF WORDS ERROR	FD	-	-	• O número de words introduzidas excede o número de words definido para a arquitectura na MIPS_PACKAGE
*	PORT NUMBER ERROR	FE	-	-	• Porta Invalida ou gama de portas de dados a ler não válido
*	UNKNOWN ERROR	FF	-	-	• Erro desconhecido na descodificação da operação

Legenda:

* Novas operações;

** Operações já existentes, apenas optimizadas em algumas situações que causavam alguma instabilidade no protocolo de comunicação.

3 - Demonstração de funcionamento do protocolo de comunicação

Após transferir a bitstream "MIPS32_Single_Cycle_BASIC.bit" para a placa DETIUA-S3, no "terminal window", da aplicação PBM, introduzir os seguintes comandos:

Simulação da execução do MIPS:

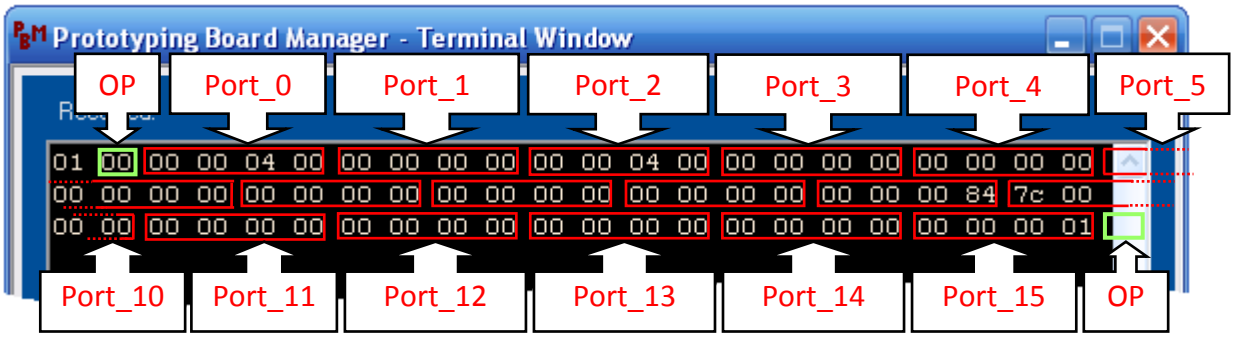
- Apagar sectores da flash correspondentes a:
  - `4 1F 80 00 1F 80 00` ➔ registos (leitura)
  - `4 1F A0 00 1F A0 00` ➔ memória de dados (leitura)
  - `4 1F C0 00 1F C0 00` ➔ `valor dos` sinais (leitura)

- Visualizar os valores do inicio de cada sector correspondente a instruções, dados e valores lidos do MIPS:
  - `5 1F 80 00 1F 80 63` ➔ Ver conteúdo do FileRegister
  - `5 1F A0 00 1F A0 63` ➔ Ver conteúdo da DataMemory
  - `5 1F C0 00 1F C0 3C` ➔ Ver valores dos sinais

- Buscar os valores do MIPS:
  - `A 00 00 00 00 00 0F` ➔ Ler as 16 portas e escrever na flash
  - `5 1F C0 00 1F C0 3F` ➔ Ler da flash os valores das 16 portas (visualiza-se em grupos de 4 bytes representando os seguintes sinais:

```
-- Output signals to interface --
Port_0 <= NextPC;
Port_1 <= PCOut;
Port_2 <= PC4;
Port_3 <= Instruction;
Port_4 <= SignExtended;
Port_5 <= BranchOffset;
Port_6 <= ALUData1;
Port_7 <= Data2;
Port_8 <= ALUResult;
Port_9 <= MemData;
Port_10 <= RegDst & Branch & MemRead & MemtoReg & ALUOp & MemWrite & ALUSrc & RegWrite & -- Control Unit
           ALUCtrlCode &   -- ALU Control Code
           Zero &      -- ALU
           BranchAndZero & -- AND
           '0' & X"0000" ; -- EMPTY BITS
Port_11 <= RegisterData;
Port_12 <= DataMemoryData;
```

Exemplo:



- Executar o MIPS para 2 instruções:
  - 8 00 00 07 00 00 00 ➔ Gera 8 ciclos de relógio
  - A 00 00 00 00 00 0F ➔ Obter os valores do datapath
  - 5 1F C0 00 1F C0 3F ➔ Ver que o MIPS encontra-se na 2 instrução PCOut = 0x04
  - 1B 00 00 0B 00 00 1F ➔ Obter de modo seguro o conteúdo do FileRegister
  - 5 1F 80 00 1F 80 7F ➔ Verificar que o registo 1 possui o valor 4

- Para efectuar um Reset ao MIPS:
  - 9 00 00 03 00 00 00 ➔ 1 CPI em RESET

- Executar todas as instruções no MIPS:
  - 8 00 00 70 00 00 00 ➔ Gera 112 ciclos de relógio
  - A 00 00 00 00 00 0F ➔ Obter os valores do datapath
  - 5 1F C0 00 1F C0 3F ➔ Ver que o MIPS encontra-se na última instrução PCOut = 0x30 e verifica-se que no sinal à saída de DATAMEM (Port_9 ← MemData), tem-se o valor pretendido da execução do trecho de código assembly, o mesmo que obtido em simulação: "A" correspondente à soma do array = { 1 + 2 + 3 +4 } em hexadecimal.
  - B 00 00 0B 00 00 1F ➔ Obter o conteúdo do FileRegister (normal)
  - 5 1F 80 00 1F 80 63 ➔ Verificar os registos possuem os valores esperados
  - C 00 00 0C 00 00 1F ➔ Obter o conteúdo da Data Memory (também pode ser efectuado no modo seguro)
  - 5 1F A0 00 1F A0 63 ➔ Verificar o conteúdo da Data Memory possui os valores esperados

- Efectuar um Reset ao MIPS:
  - 9 00 00 03 00 00 00 ➔ Activa o RESET e gera 1 CPI. Após o este comando, pode-se efectuar novamente o conjunto de comandos desta folha, obtendo-se os mesmos resultados.

Simulação da transferência de instruções e dados para o MIPS:

- Apagar os sectores de memória Flash correspondentes a instruções e dados:
  - 4 1E 00 00 1E 00 00 ➔ instruções (inicialização)
  - 4 1F 00 00 1F 00 00 ➔ memoria (inicialização)

- Verificar se os sectores estão apagados:
  - 5 1E 00 00 1E 00 3C ➔ Ver sector de instruções (todos os bytes a 0xF)
  - 5 1F 00 00 1F 00 3C ➔ Ver sector de dados da memória (todos os bytes a 0xF)

- Escrever 3 instruções na flash:
  - 6 1E 00 00 1E 00 0B ➔ Escrever 12 bytes a partir do endereço 1f8000
  - 00 00 00 00 ➔ NOP
  - 8C 01 00 00 ➔ LW r1, 0(r0)
  - 10 00 FF FF ➔ BEQ r0, r0, -1 (salta para ele próprio infinitamente)

- Verificar se as instruções foram bem escritas:
  - 5 1E 00 00 1E 00 3F ➔ Verificar se os primeiros bytes representam as instruções enviadas

- Escrever 32bit de dados na flash:
  - `6 1F 00 00 1F 00 03` ➔ Escrever 4 bytes a partir do sector de dados
  - `1F 2F 3F 4F` ➔ Substituirá o valor do primeiro endereço da DataMemory

- Verificar se as instruções foram bem escritas:
  - `5 1F 00 00 1F 00 3F` ➔ Verificar se os primeiros bytes representam as instruções enviadas

- Carregar um dado e duas instruções:
  - `1F 00 00 00 00 00 3F` ➔ Apagar Dados de DATAMEM (todos os dados = 0x0000 0000)
  - `E 00 00 00 00 00 00` ➔ Ler um dado para DATAMEM
  - `C 00 00 0C 00 00 1F` ➔ Obter o conteúdo da Data Memory (também pode ser efectuado no modo seguro)
  - `5 1F A0 00 1F A0 63` ➔ Verificar o conteúdo da Data Memory possui os valores esperados
  - `F 00 00 00 00 00 3F` ➔ Apagar INSTMEM (todas as instruções passam a ser "nop")
  - `D 00 00 00 00 00 02` ➔ Ler 3 instruções para INSTMEM
  - `9 00 00 03 00 00 00` ➔ 1 CPI em Reset para inicializar
  - `A 00 00 00 00 00 0F` ➔ Ler sinais do datapath
  - `5 1F C0 00 1F C0 3F` ➔ Visualizar os dados (primeira instrução é um nop)

- Verificar dados de cada ciclo:
  - `8 00 00 07 00 00 00` ➔ Efectuar 2 CPI ao MIPS Single Cycle
  - `A 00 00 00 00 00 0F` ➔ Ler sinais do datapath
  - `5 1F C0 00 1F C0 3F` ➔ Visualizar dados e ver que o LW lê de DATAMEM o valor introduzido "1F 2F 3F 4F".
  - **Repetir últimos 3 comandos** ➔ Visualizar os dados e verificar que a 2 instrução agora é o beq introduzido anteriormente e que se efectuar mais ciclos nunca passará dessa instrução, ou seja, NextPC e PCOut = x08 sucessivamente.

Outras sequências de operações são possíveis de execução. As anteriores servem apenas para demonstrar e testar as funcionalidades implementadas.

4 – Diagramas temporais dos sinais gerados pelo protocolo de comunicação

No intuído de ajudar o utilizador a conhecer que sinais são produzidos pela interface implementada para o processador, de seguida são demonstrados alguns diagramas temporais dos sinais gerados. Na figura 1 encontra a implementação do processador MIPS Single Cycle em conjunção com o protocolo de comunicação e foi sobre a qual que obteve os próximos diagramas.

NOTA: Os sinais apresentados não demonstram a interactividade com a memória flash ou o módulo USB.

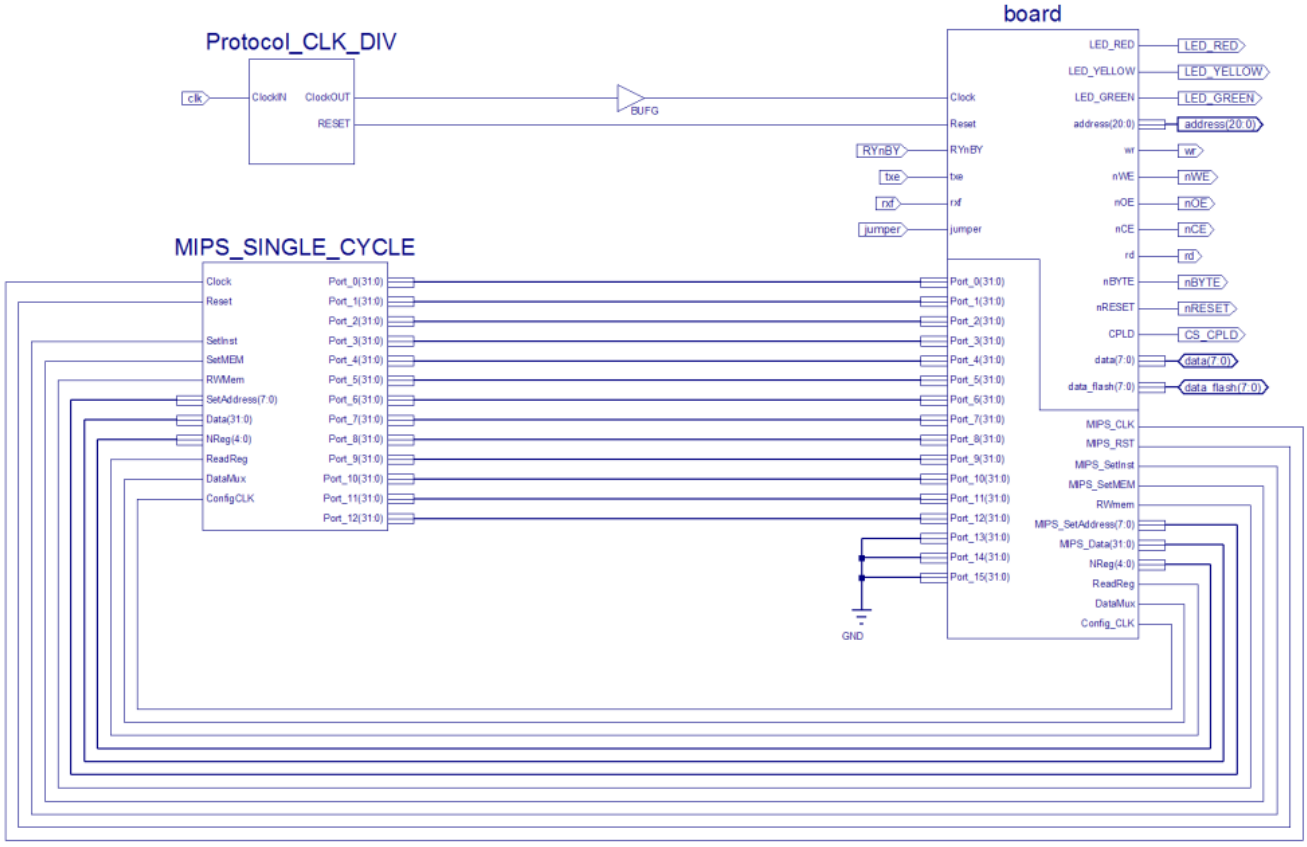


Figura 1 – Diagrama de blocos do projecto MIPS32 Single Cycle Basic e Advanced

MIPS GET DATA

Não é apresentado o diagrama temporal da evolução dos sinais gerados para a operação MIPS GET DATA, visto que é uma operação interna em que unicamente lê os valores de todos sinais ligados às portas dedicadas (PORT_n de 32bit cada) e guarda-os na memória flash para posterior leitura. Nesta operação não são gerados sinais que influenciem o normal funcionamento do processador.

MIPS CYCLE

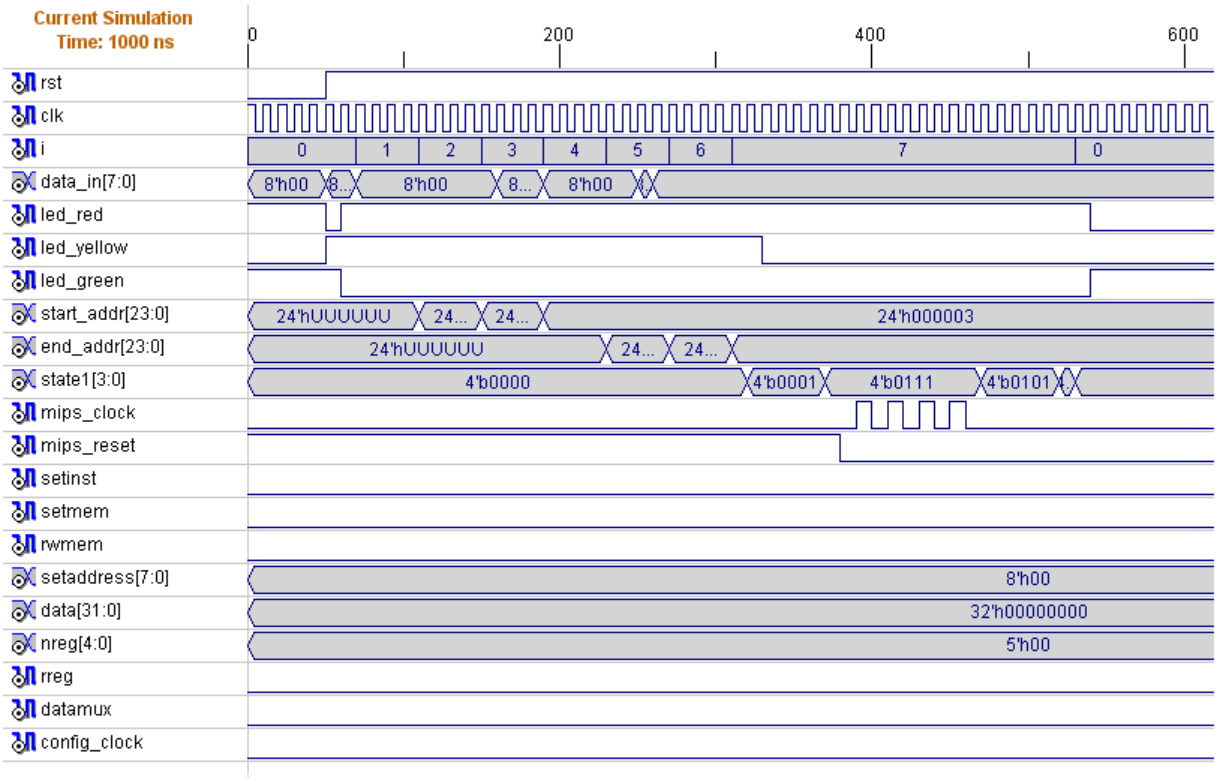


Figura 2 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS CYCLE com os parâmetros CPI=4 e CLKHZ = 0;

MIPS RESET

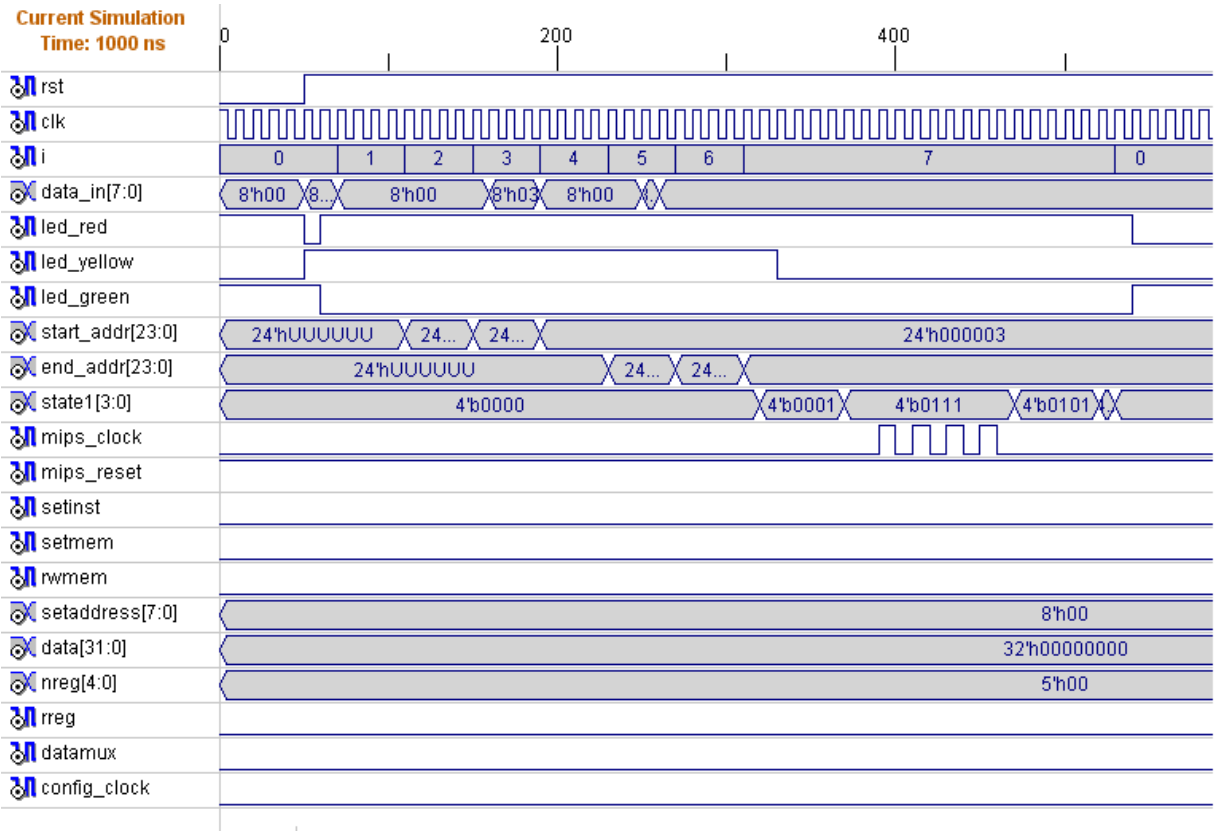


Figura 3 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS RESET com os parâmetros CPI=4 e CLKHZ = 0;

MIPS READ FILE REGISTER

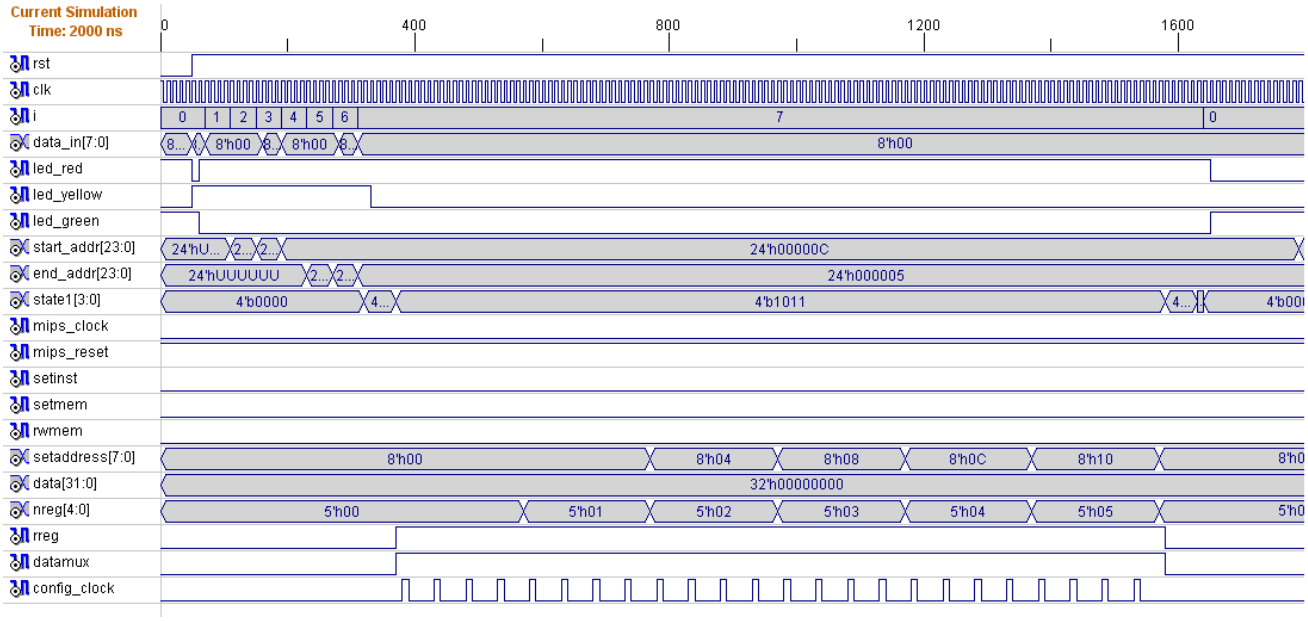


Figura 4 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ FILE REGISTER com os parâmetros PORT=11 e Nº Registos = 5;

MIPS READ FILE REGISTER (Safe mode)

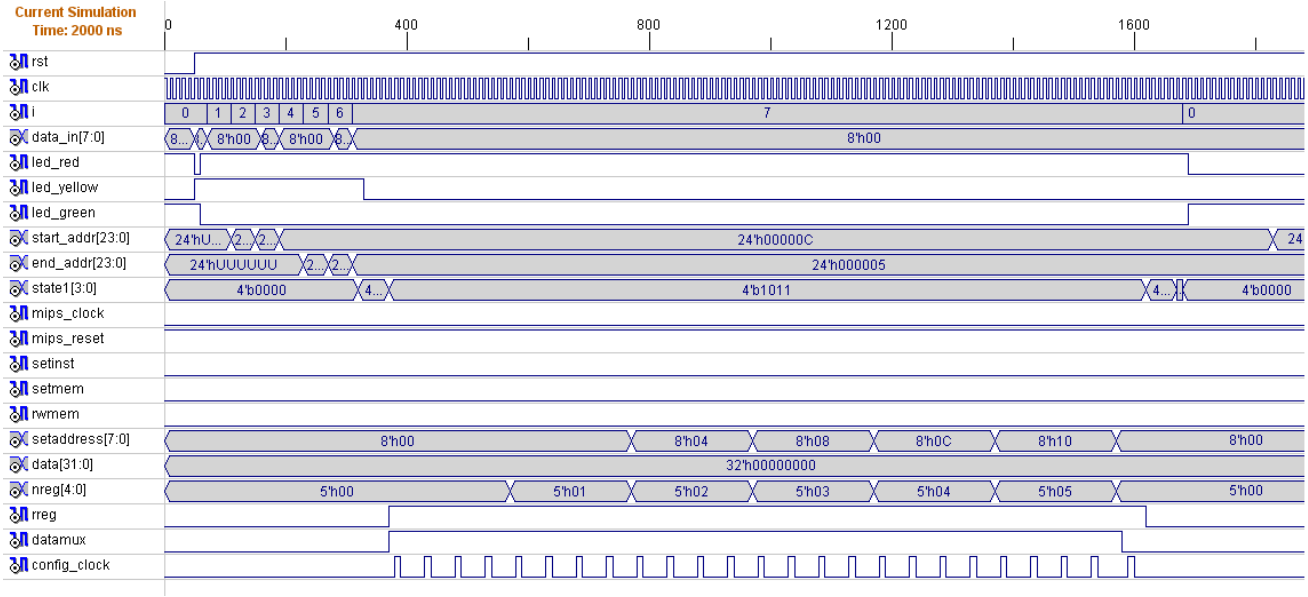


Figura 5 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ FILE REGISTER (Safe) com os parâmetros PORT=11 e Nº Registos = 5;

MIPS READ DATA MEMORY

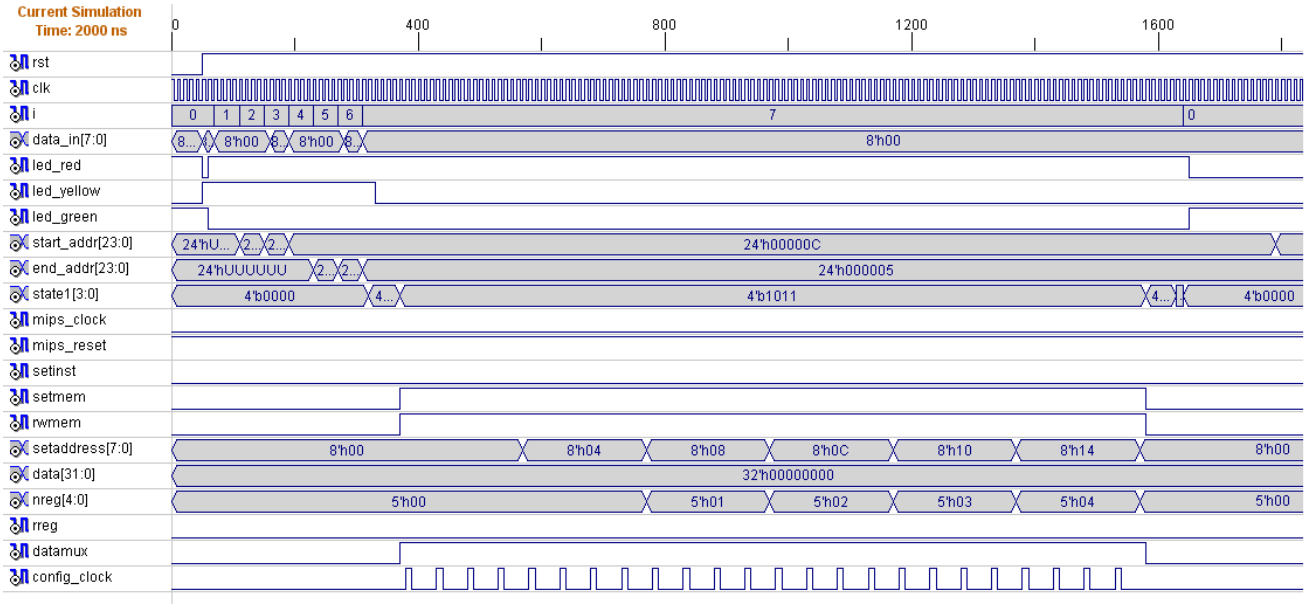


Figura 6 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ DATA MEMORY com os parâmetros PORT=12 e Nº Words = 5;

MIPS READ DATA MEMORY (Safe mode)

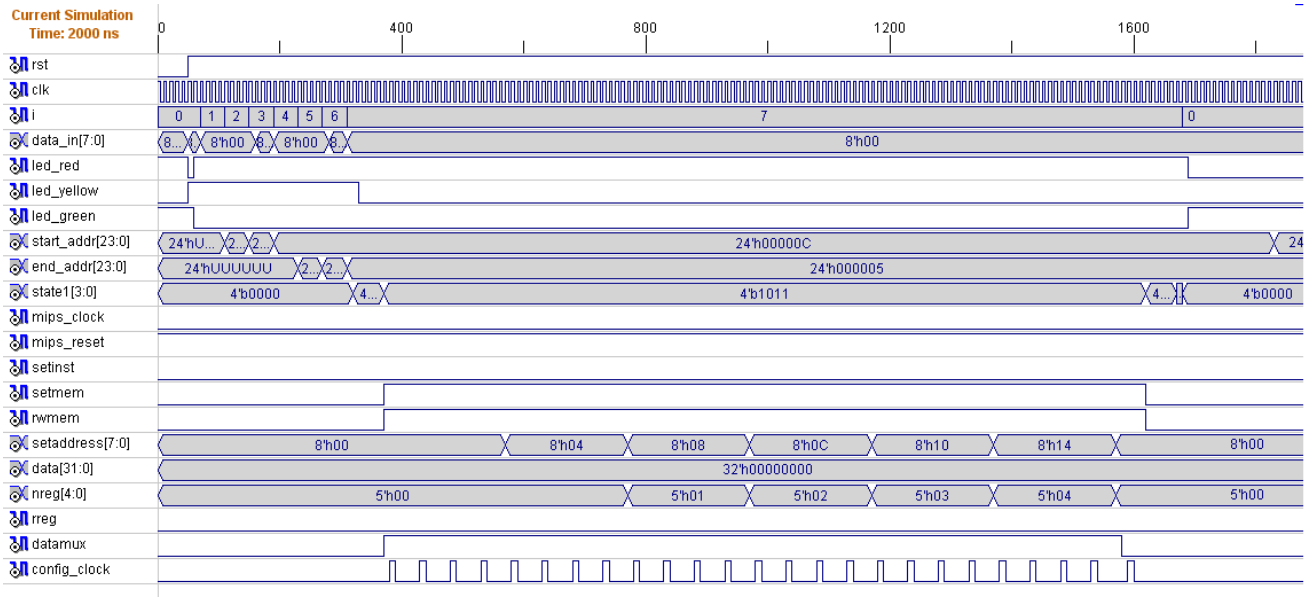


Figura 7 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS READ DATA MEMORY (SAFE) com os parâmetros PORT=12 e Nº Words = 5;

MIPS SET INSTRUCTION MEMORY

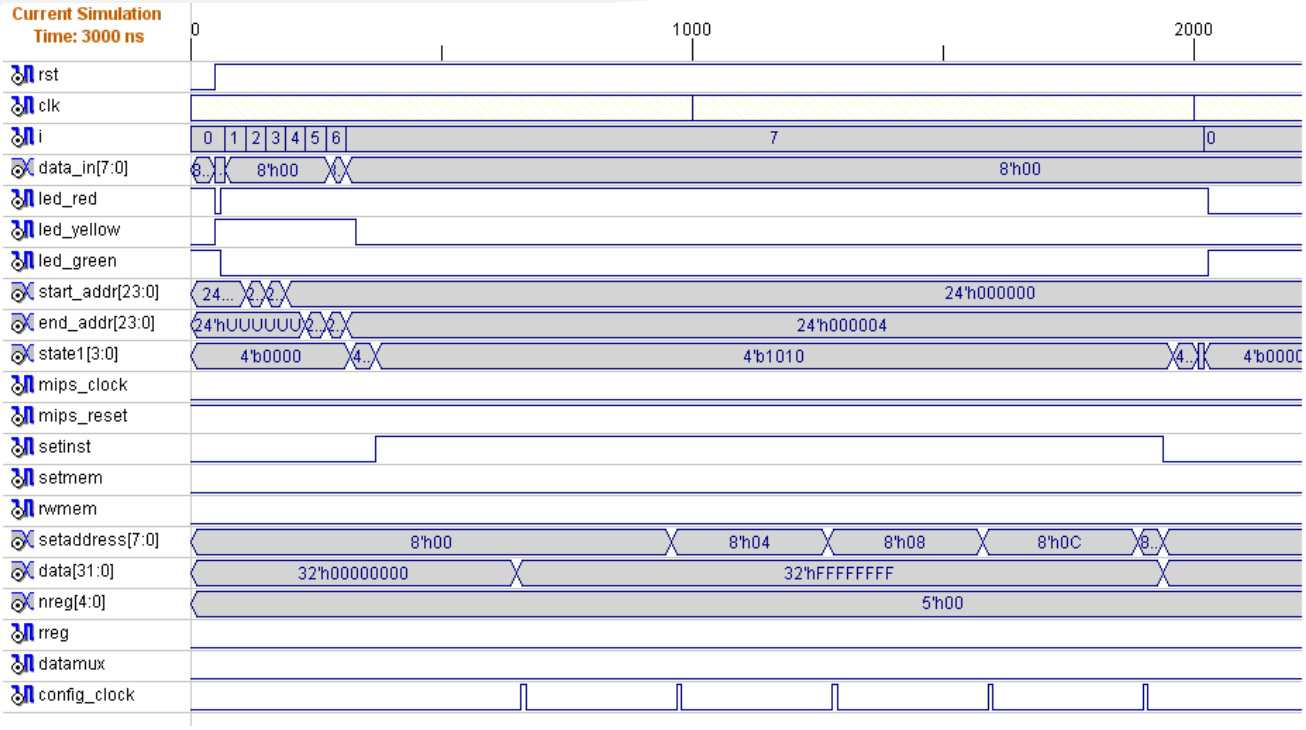


Figura 8 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS SET INSTRUCTION MEMORY com os parâmetros Nº Words = 4;

MIPS SET DATA MEMORY

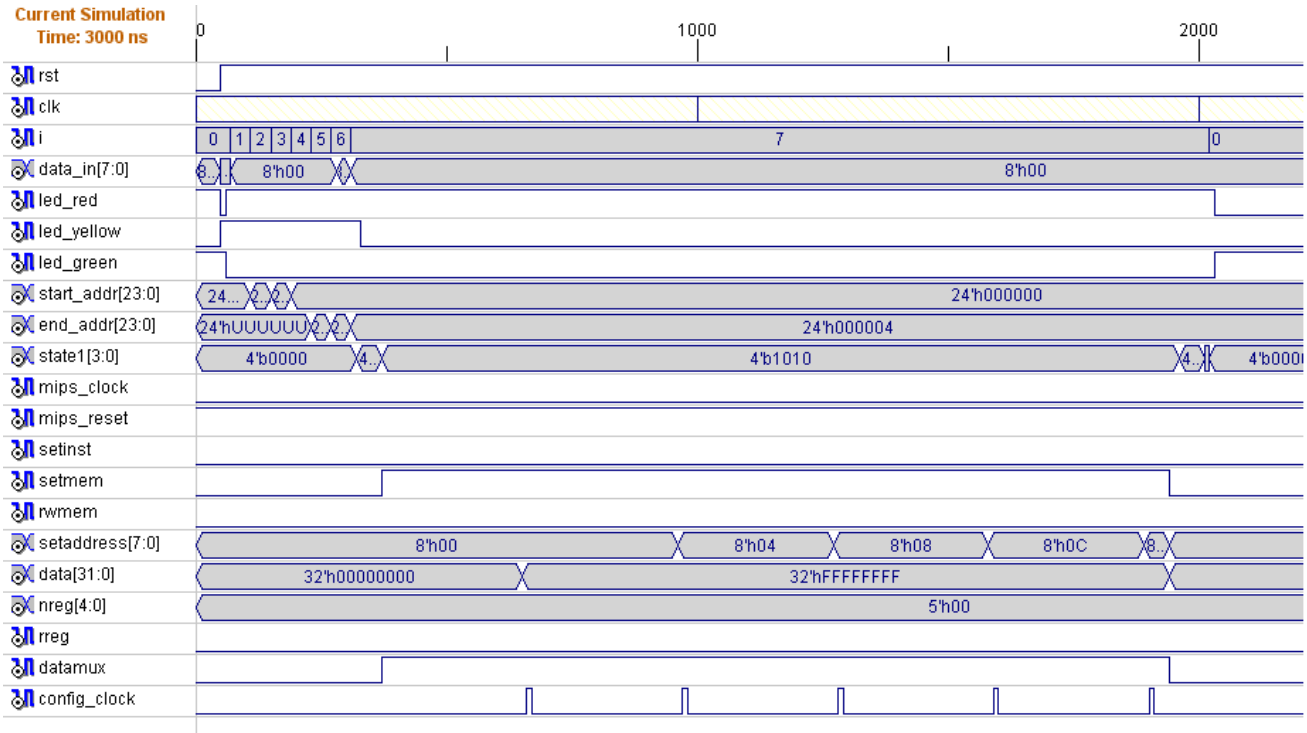


Figura 9 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS SET DATA MEMORY com os parâmetros Nº Words = 4;

MIPS ERASE INSTRUCTION MEMORY

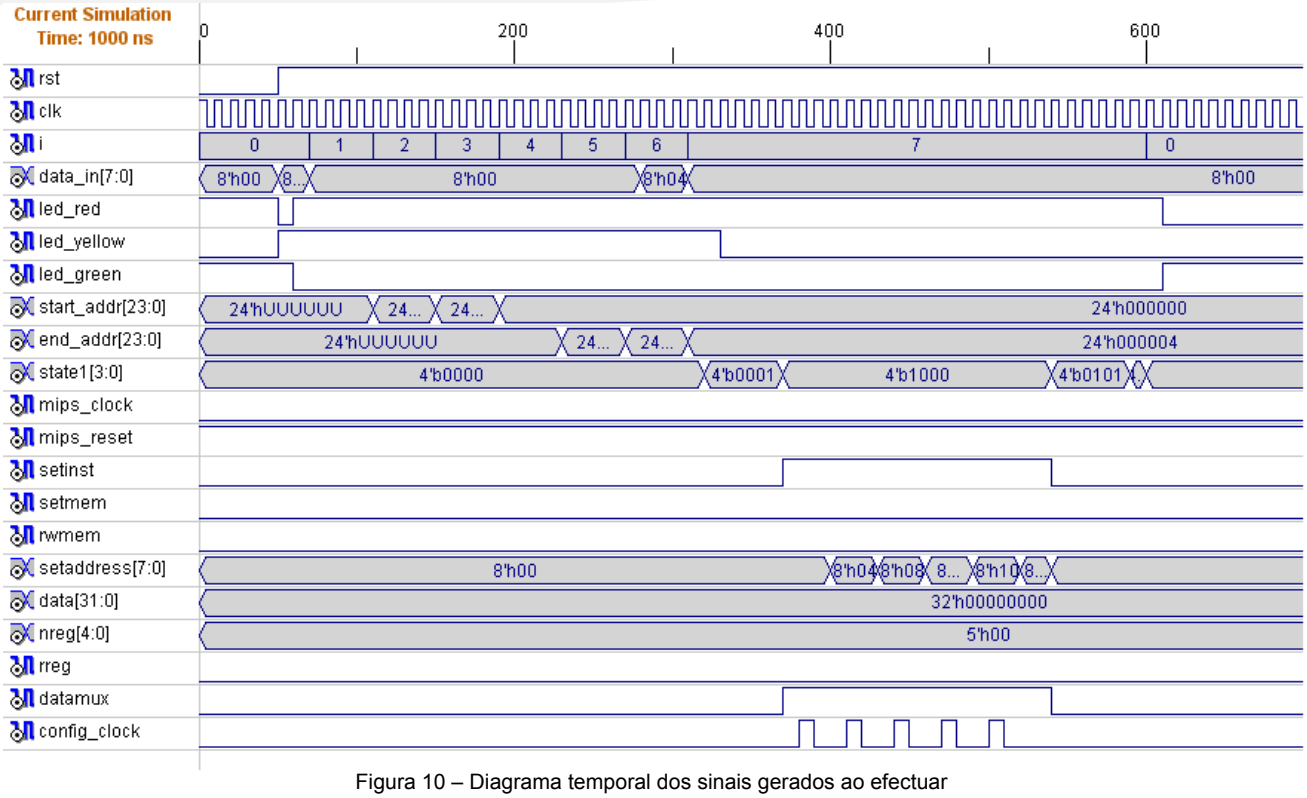


Figura 10 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS ERASE INSTUCTION MEMORY com os parâmetros Nº Words = 4;

MIPS ERASE DATA MEMORY

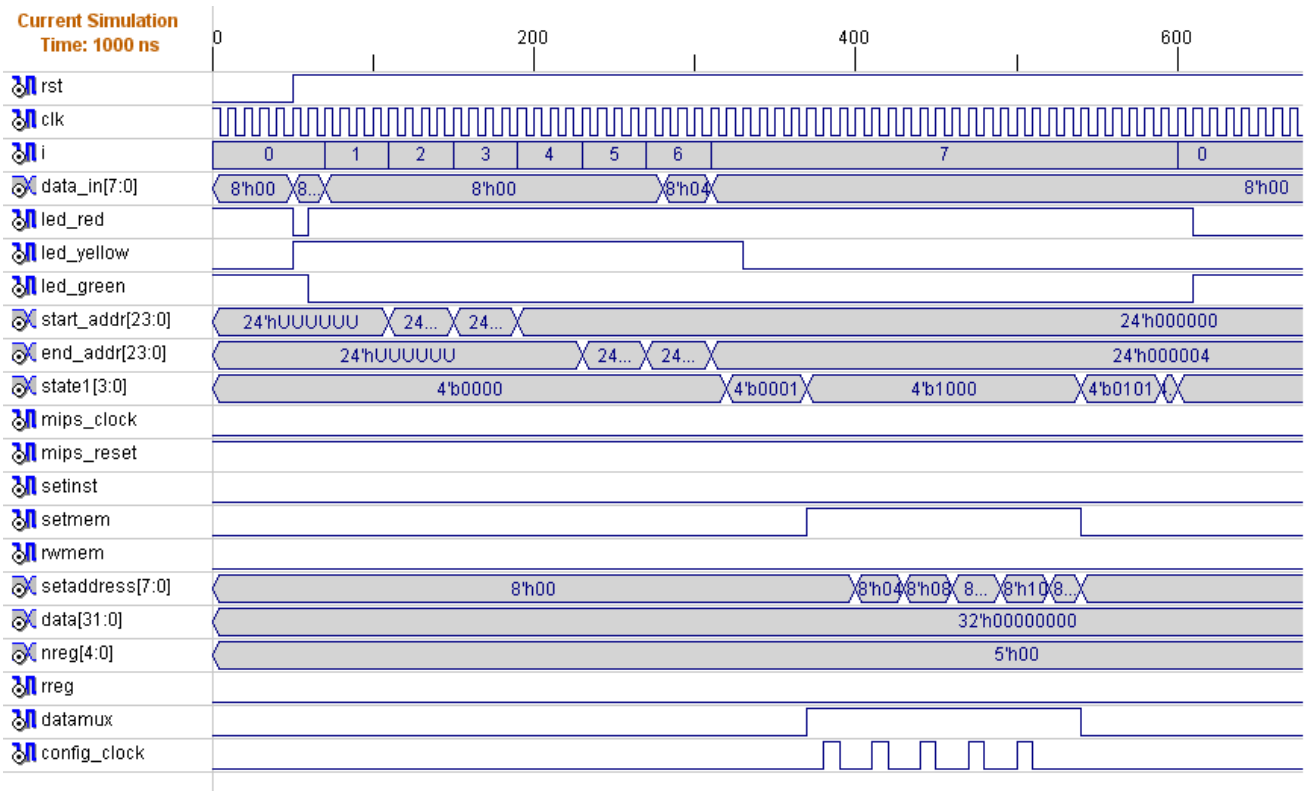


Figura 11 – Diagrama temporal dos sinais gerados ao efectuar
o comando MIPS ERASE DATA MEMORY com os parâmetros Nº Words = 4;